ط à

Cisco Cookbook

By Ian J. Brown, Kevin Dooley

Publisher: O'Reilly
Pub Date: July 2003
ISBN: 0-596-00367-6

Top

Printed in the United States of America.

Published by O'Reilly & Associates, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

O'Reilly & Associates books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (http://safari.oreilly.com). For more information, contact our corporate/institutional sales department: (800) 998-9938 or corporate@oreilly.com.

Top

Preface

Cisco routers are nearly ubiquitous in IP networks. They are extremely flexible and reliable devices, and the number and variety of features grows with each new release of the Internetwork Operating System (IOS). While Cisco Press and several other publishers supply excellent documentation of router features both online and in a variety of books, knowing when, why, and how to use these features is sometimes difficult. There are often many different ways to solve any given networking problem using Cisco devices, and some solutions are clearly more effective than others.

The two immediate questions facing any network engineer are: Which of the many potential solutions is the most appropriate for a particular situation? and, Once you have decided to use a particular feature, how should you implement it? Unfortunately, the feature documentation describing a particular command or feature frequently does very little to answer either of these questions.

Everybody who has worked with Cisco routers for any length of time has had to ask their friends and co-workers for example router configuration files that show how to solve a common problem. A good working configuration example can often save huge amounts of time and minimize the frustration that sometimes comes with implementing a feature that you've never used before.

Cisco Cookbook is not intended to replace the detailed feature documentation included in books such as Cisco IOS in a Nutshell (O'Reilly) or information available on Cisco's web site (http://www.cisco.com). While we don't have the space to provide details about how particular protocols actually work, you can find this information in the Internet Engineering Task Force (IETF) Request for Comment (RFC) documents (located at http://www.ietf.org/rfc.html) and a wide variety of books.

Instead, this book is a complement to those sources of information. They will tell you what a routing protocol is, how it works, and which command turns it on. Cisco Cookbook will help you select the right routing protocol and configure it in the most efficient way for your network.

This book includes a collection of sample router configurations and scripts that we have found useful in real-world networks. It also includes, wherever possible, our advice on what features to use in which situations, and how to use them most effectively. There are many common mistakes that we have seen before, and we want to help you to avoid making them.

All of the recipes in this book should work with IOS levels 11.3, 12.0, 12.1, 12.2, and 12.3. And, except where noted, they should run on any Cisco router platform. We have indicated when we use features that are only available with certain release levels or code sets, and in some cases offered workarounds for older versions. It is also important to remember that most of the recipes will work not only with

Cisco routers, but also with any Catalyst switches that run IOS (but unfortunately not CatOS switches). In particular, all of the recipes that pertain to AAA, security, syslog, and SNMP should work well on these devices.

We welcome feedback from our readers. If you have comments, suggestions or ideas for other recipes, please let us know. If there are future editions of the Cisco Cookbook, we will include any suggestions that we think are especially useful. You can reach us at: kevind@manageablenetworks.com or ijbrown@hotmail.com.

Top

Organization

As the name suggests, Cisco Cookbook is organized as a series of recipes. Each recipe begins with a problem statement that describes a common situation that you might face. After each problem statement is a brief solution that shows a sample router configuration or script that you can use to resolve that particular problem. A discussion section then describes the solution, how it works, and when you should or should not use it.

We have tried to construct the recipes so that you should be able to turn directly to the one that addresses your specific problem and find a useful solution without needing to read the entire book. If the solution includes terms or concepts that you are not familiar with, the chapter introductions should help bridge the gap. Many recipes refer to other recipes or chapters that discuss related topics. We have also included a variety of references to other sources in case you need more background information on a particular subject.

The chapters are organized by the feature or protocol discussed. If you are looking for information on a particular feature such as NAT, NTP, or SNMP, you can turn to that chapter and find a variety of related recipes. Most chapters list basic problems first, and any unusual or complicated situations last. But there are some exceptions to this, where we have opted instead to group related recipes together.

Top

What's in This Book

The first four chapters cover what would be considered essential system administration functions if a router were a server. Chapter 1 covers router configuration and file management issues. In Chapter 2, we turn to useful router management tricks such as command aliases, using CDP and DNS, tuning buffers, and creating exception dumps. This chapter ends with a set of four scripts that generate various useful reports to help you manage your routers. Chapter 3 discusses user access and privileges on the router. Chapter 4 extends this discussion to using TACACS+ to provide centralized management of user access to your routers.

The next five chapters cover various aspects of IP routing. Chapter 5 looks at IP routing in general, including static routes and administrative distances. In Chapter 6, we focus on RIP, including both Versions 1 and 2. Chapter 7 looks at EIGRP, and Chapter 8 at OSPF. In Chapter 9, we discuss the BGP protocol, which controls all IP routing through the backbone of the Internet.

The remaining chapters all cover separate topics. We look at the popular Frame Relay WAN protocol in Chapter 10.

Chapter 11 discusses queuing and congestion. This chapter also examines various IP Quality of Service issues.

In Chapter 12, we look at IP tunnels and VPNs. This chapter includes a discussion of Cisco's IPSec implementation.

We turn to issues related to dial backup in Chapter 13.

In Chapter 14, we look at time. We include a relatively detailed discussion of the NTP protocol, which you can use to synchronize the clocks of all of your routers. You can then use them as time sources for other equipment, including application servers on your network.

Chapter 15 is primarily concerned with configuring the DLSw protocol. It also looks at SNA and SDLC protocols, which are often carried over IP networks using DLSw.

In Chapter 16, we show how to configure several of the most popular interface types on a Cisco router.

Chapter 17 and Chapter 18 look at the closely related issues of network management and logging. In Chapter 17, we discuss SNMP in particular. This chapter includes several router configuration examples to use with SNMP, as well as a number of scripts that you can use to help manage your Cisco equipment. Chapter 18 looks at issues related to managing the router's event logs, as well how to use the syslog protocol to send these log messages to a central server.

It's impossible to do much on a Cisco router without having a good understanding of access lists. There are several different kinds of access lists, and Chapter 19 shows several useful and interesting applications of the various IP-specific access lists.

In Chapter 20, we look at DHCP. Routers usually just act as DHCP proxy devices, but we also show how to use the router as a DHCP server, or even as a client.

Chapter 21 talks about NAT, which allows you to use private IP addresses and resolve conflicting address ranges between networks.

One of the best ways to build a fault tolerant LAN is to configure two or more routers to share a single IP address using HSRP. We show several different HSRP configurations in Chapter 22.

In Chapter 23, we look at how to implement multicast routing functionality on a Cisco router.

We also include two appendixes. Appendix A discusses the various external software tools that we use throughout the book, and shows how to obtain your own copies of these packages. Appendix B gives some helpful background on IP Quality of Service and the various queueing algorithms that you can use on Cisco routers.

Top

Conventions

The following formatting conventions are used throughout this book:

- Italic is used for commands, file names, directories, script variables, keywords, emphasis, technical terms, and Internet domain names.
- `Constant width` is used for code sections, interface names, and IP addresses.
- `Constant width italic` is used for replaceable text.
- **`Constant width bold`** is used for user input and emphasis within code.
- *`Constant width bold italic`* is used to highlight replaceable items within code.

Top

Comments and Questions

Please address comments and questions about this book to the publisher:

O'Reilly & Associates, Inc.
1005 Gravenstein Highway North
Sebastopol, CA 95472
(800) 998-9938 (in the United States or Canada)
(707) 829-0515 (international/local)
(707) 829-0104 (fax)

There is a web page for this book, which lists errata, examples, or any additional information. You can access this page at:

http://www.oreilly.com/catalog/ciscockbk/

To comment or ask technical questions about this book, send email to:

bookquestions@oreilly.com

For more information about books, conferences, Resource Centers, and the O'Reilly Network, see the O'Reilly web site at:

http://www.oreilly.com

Top

Acknowledgments

Writing this book was a huge project, and we are grateful that so many people helped us in different ways. We want to extend particularly large thanks to John Karek for helping to set up the lab environment that we used to testing recipes; Jackman Chan, who ran some of the more obscure and difficult debugging traces for us; and to David Close of Cisco Canada, who very generously loaned us equipment at a critical phase.

Everybody at O'Reilly was great to work with. We particularly appreciate the hard work of our editors, Jim Sumser, Mike Loukides, and Phil Dangler. They encouraged us as we wrote. And, when we were done writing, they wrestled the results into something we all could be proud of. Jessamyn Read did a great job with the figures, and we were completely thrilled with Ellie Volckhausen's cover art.

We had three technical reviewers for this book, and they each made a huge contribution by both pointing out our errors and making useful suggestions on how to present the material. Peter Rybaczyk and Ravi Malhotra both showed incredible breadth and depth of knowledge of Cisco routers and networking in general, as well as offering help with the overall structure and flow of the book. And we leaned very heavily on Iljitsch van Beijnum for help with the BGP chapter.

Kevin Dooley

There is a lot more to writing a book than just the writing. I would like to thank Sherry Biscope, who has now survived my writing two books. And, midway through this one, she almost crazily agreed to marry me. But she did far more than merely survive. She encouraged and prodded, she made time and space for this book, and she barely complained at all when piles of books, papers, routers, and cables took over the living room. And thanks also to Ginger the beagle who slept in the big comfy chair throughout the writing of this book, always within petting distance, usually very forgiving of the delays in walks and dinner time.

Ian J. Brown

I would like to thank my beautiful wife, Lisa, who supported me unconditionally throughout this project, and in doing so, became the sole caregiver to our young children. Without your assistance and encouragement, this book would never have happened. Special thanks also to my son, Ethan, and daughter, Darby, who endured many evenings without a father. You mean the world to me and I will love you always and forever. I would also like to thank Alan Morewood, who inspired more than one section of this book.

Top

Chapter 1. Router Configuration and File Management

Top

Introduction

You can think of a Cisco router as a special-purpose computer. It has its own operating system, which is called the Internetwork Operating System (IOS), as well as files and filesystems. So we'll start with a discussion of the basic system administration functions that a router engineer must perform. This includes managing your router's filesystems, upgrading the operating system, doing backups, and restoring the system configuration.

Cisco routers use flash memory, rather than disks, for storing information. Flash storage media is significantly more expensive and slower than disk storage, but the amount of storage needed to run a router is relatively small compared to the amount needed to run a general-purpose computer. Flash also has the important benefit that it tends to be more reliable than disk storage.

Flash storage is similar to Random Access Memory (RAM), but it doesn't need power to retain information, so it is called non-volatile. And, unlike Read Only Memory (ROM), you can erase and rewrite flash easily. There are other types of non-volatile solid state storage, such as Erasable Programmable Read Only Memory (EPROM) and Electronically Erasable Programmable Read Only Memory (EEPROM). EPROM is not suitable for routers because it generally requires an external device such as an ultraviolet light shone through a window on the chip to erase it. EEPROM, on the other hand, can be erased by simply sending an erase signal to the chip. But there is a key difference between EEPROM and flash memory: when you erase something from an EEPROM device, you must erase the entire device, while flash devices allow selective deletion of parts of the medium.

This is an important feature for routers, because you don't always want to erase the entire storage medium in order to erase a single file. In Recipe 1.11 and Recipe 1.12, we discuss ways to erase single files on some types of routers, depending on the type of filesystem used.

There are at least two main pieces of non-volatile storage in a Cisco router. The router's configuration information is stored in a device called the Non-Volatile RAM (NVRAM), and the IOS images are stored in a device called the flash (lowercase). It's important to keep these names straight because, of course, all Flash memory is non-volatile RAM. And, in fact, most routers use Flash technology for their NVRAM. So it's easy to get confused by the terms.

On most Cisco routers, the NVRAM area is somewhere between 16 and 256Kb, depending on the size and function of the router. Larger routers are expected to have larger configuration files, so they need more NVRAM. The flash device, on the other hand, is usually upgradeable, and can be anywhere from a few megabytes to hundreds of megabytes.

We often talk about a router's configuration file, but there are actually two important configuration files on any router. There is the configuration file that describes the current running state of the router,

which is called the *running-config*. Then, there is the configuration file that the router uses to boot, which is called the *startup-config*. Only the *startup-config* is stored in NVRAM, so it is important to periodically check that the version of the configuration in the NVRAM is synchronized with the version that the router is currently running. Otherwise you could get a surprise from ancient history the next time the router reboots. You can synchronize the two configuration files by simply copying the *running-config* onto the *startup-config* file:

```
Router1#copy running-config startup-config
```

Many Cisco engineers, including the authors, still use the old-fashioned version of this command out of force of habit:

```
Router1#write memory
```

However, this command is not only deprecated, it's also less descriptive of what the router is doing.

The router uses the larger flash storage device for holding the operating system, or IOS. Unlike the operating systems on most computers, the IOS is a single file containing all of the features and functions available on the router. You can obtain the IOS image files from Cisco on CD or, if you have an account on their system, you can download IOS files from the Cisco web site using FTP.

Most of the examples throughout this book assume that you have IOS Version 12. However, many of the features we discuss are also available in earlier versions. Although there may be slight syntax changes, we expect that Cisco will continue to support all of the features we describe well into the future. It is important to be flexible because if you work with Cisco routers a lot, you will encounter a large variety of different IOS versions, with various subtle differences. Unfortunately, some of these subtle differences are actually bugs. Cisco offers a detailed bug tracking system on their web site for registered users.

There are several important things to consider when you go to change the IOS version on a router. First is the feature set. For each IOS release, Cisco produces several different versions. They usually offer an Enterprise Feature Set, which includes all of the different feature options available at a given time. Because the IOS is a monolithic file containing all features and all commands, the Enterprise IOS files are usually quite large. The Enterprise version is generally much more expensive than the various stripped-down versions.

The simplest IOS version is usually the IP Only Feature Set. As the name suggests, this includes only TCP/IP based functionality. In most networks, you will find that the IP Only Feature Set is more than sufficient. In fact, almost all of the recipes in this book will work with the IP Only version of IOS.

If you require other protocols such as IPX or AppleTalk, Cisco produces an IOS Feature Set called Desktop that contains these protocols. They also offer several other important variations such as IP Plus, IP Plus IPSec 56, IP Plus IPSec 3DES, and so forth. The contents of these different versions (and even their names to some extent) vary from release to release. We encourage you to consult Cisco's feature

matrixes to ensure that the features you need are in the IOS version that you have.

One of the most important considerations with any IOS release is whether you have sufficient RAM and Flash memory to support the new version. You can see how much storage your router has by looking at the output of the *show version* command.

The other important thing to remember about IOS images on Cisco routers is that every router has a fallback image located in the router's ROM. This IOS image cannot be changed or upgraded without physically replacing the ROM chips in the router.

The router's ROM contains three items: the power on self test (POST), the bootstrap program, and a limited version of the router's operating system. The router uses the bootstrap program while booting. The IOS image in ROM is usually an extremely stripped-down version that doesn't support many common features (routing protocols, for example). In the normal boot cycle, the router will first load the POST, then the bootstrap program followed by the appropriate IOS image. Please refer to Recipe 1.7 for more information about booting from different IOS files.

Recipe 1.7 also shows how to adjust the configuration register values. These values set a variety of boot options, and even allow you to force the router to stop its boot process before loading the IOS. This can be useful if the IOS image is corrupted, or if you need to do password recovery.

Top

Recipe 1.1 Configuring the Router via TFTP

1.1.1 Problem

You want to load configuration commands via the Trivial File Transfer Protocol (TFTP).

1.1.2 Solution

You can use the *copy tftp:* command to configure the router via the TFTP:

```
Router1#copy tftp://172.25.1.1/NEWCONFIG  running-config
Destination filename [running-config]? <enter>
Accessing  tftp://172.25.1.1/NEWCONFIG...
Loading NEWCONFIG from 172.25.1.1 (via FastEthernet0/0.1): !
[OK - 24 bytes]

24 bytes copied in 0.192 secs (125 bytes/sec)
Router1#
```

> IOS versions before 12.0 used the command *configure network*. This command is still available in more recent versions, but it is now deprecated and may not continue to be available in the future.

1.1.3 Discussion

Generally, most people configure their routers using Telnet and the *configure terminal* command. For large configuration changes, people tend to resort to cutting and pasting a large set of commands. While this method works, it is inefficient and slow, particularly if you have to configure large numbers of routers. When you use TFTP to download a large set of configuration commands, the router doesn't need to echo each character to your screen, which reduces the overhead and increases the speed.

In our example, we configured the router by making it download the file called *NEWCONFIG* from the server at `172.25.1.1` using TFTP. The router copies the entire file via TFTP before entering the commands into the running configuration. This is extremely useful because using some commands in the middle of a configuration could disrupt your access to the router—but the rest of the commands might fix the problem. If you tried to enter them manually using Telnet and *configure terminal*, you would simply lock yourself out of the router. A typical example of this problem happens when you replace an active access list. When you enter the first line, the router puts an implicit deny all at the

end, which could break your session. However, entering commands using TFTP avoids this problem.

The last line of any configuration file that you copy into the router like this should be the *end* command. This lets the router know that it has reached the end of the file. If you don't do this, the router will still accept all of the commands normally, but it will put the following error into its logs:

```
Jan 19 11:26:38: %PARSER-4-BADCFG: Unexpected end of configuration file.
```

If you have the *end* command in your configuration file, seeing this message will tell you that the router didn't get all of the configuration commands. But if you don't terminate the file properly, it's impossible to tell if the transfer was successful.

Instead of TFTP, you can use the FTP protocol to download configuration files. FTP has a number of advantages over TFTP in terms of reliability and security. Recipe 1.14 shows how to load configuration commands using FTP instead of TFTP.

1.1.4 See Also

Recipe 1.14

Top

Recipe 1.2 Saving Router Configuration to Server

1.2.1 Problem

You want to store a backup copy of your router's configuration on a TFTP server.

1.2.2 Solution

This example shows how to use TFTP to upload a copy of the router's active configuration to a remote server:

```
Freebsd% touch /tftpboot/router1-confg
Freebsd% chmod 666 /tftpboot/router1-confg
Freebsd% telnet Router1
Trying 172.25.1.5...
Connected to Router1.
Escape character is '^]'.

User Access Verification

Password: <vtypassword>

Router1>en
Password: <enablepassword>
Router1#copy running-config tftp://172.25.1.1/router1-confg
Address or name of remote host [172.25.1.1]? <enter>
Destination filename [router1-confg]? <enter>
!!!
9640 bytes copied in 3.956 secs (2437 bytes/sec)
Router1#
```

1.2.3 Discussion

We cannot overstress the importance of making regular backups of your router configuration files, and keeping copies of these files in a safe place. If a serious failure damages a router's hardware or software, your configuration will be destroyed. Anybody who has had to reconstruct a complex router configuration file from memory can tell you how difficult and stressful this task is! But, if you have a backup of the last working configuration file, you can usually get a router working again within minutes of fixing any hardware problems.

Typical Mean Time Between Failure (MTBF) estimates for Cisco routers tend to be about 16 years. This

sounds like a long time, but in a large network it means that you can expect to see a few failures every year. Unfortunately, human errors resulting in complete or partial loss of the configuration file are far more common than device failures.

In the example, we created an empty backup configuration file on the TFTP server, then instructed the router to send its running configuration to this server. It is important to adjust the file permissions with the Unix *chmod* command. The transfer will fail if the configuration file is not *world writable*. We highly recommend moving the configuration files out of the TFTP directory to ensure that the file isn't read by unauthorized users, or accidentally overwritten.

Reading files located in the TFTP directory is trivial, because the TFTP program needs this directory to be both *world readable* and *world writeable*. Since router configuration files contain passwords and IP addresses, you should take steps to protect these files as much as possible. In fact, you don't even need to be logged into the TFTP server to read these files. In the following example, we are able to access the TFTP server and read a router configuration file from another router:

```
Router1#more tftp://172.25.1.1/router1-confg
!
! Last configuration change at 11:23:59 EST Sat Jan 11 2003 by ijbrown
! NVRAM config last updated at 00:37:16 EST Sat Jan 11 2003 by ijbrown
!
version 12.2
service tcp-keepalives-in
service timestamps debug datetime msec
service timestamps log datetime localtime
service password-encryption
!
hostname Router1
<removed for brevity>
```

As you can see, any files left in the TFTP directory can be easily viewed or even deliberately corrupted. TFTP is notoriously insecure, so we recommend using care whenever you work with this protocol.

Recipe 1.18 provides an automated script that gathers the configuration files for a list of routers on a nightly basis and stores these files for 30 days, by default.

1.2.4 See Also

Recipe 1.14; Recipe 1.18

Top

Recipe 1.3 Booting the Router Using a Remote Configuration File

1.3.1 Problem

You want to boot the router using an alternate configuration.

1.3.2 Solution

The following set of commands allows you to automatically load a configuration file located on a remote TFTP server when the router boots:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#service config
Router1(config)#boot network tftp Network-auto 172.25.1.1
Router1(config)#boot host tftp Router8-auto 172.25.1.1
Router1(config)#end
Router1#
```

1.3.3 Discussion

By default, when the router reloads, it will read the configuration information from a file in its NVRAM. Cisco commonly refers to this file as the startup configuration file. However, you can configure the router to load all or part of its configuration from a remote server via TFTP. This feature does not prevent the router from loading its startup configuration from NVRAM. In fact, the router will load its local startup file before proceeding to the TFTP server files.

Uses for this feature vary, although most people who implement it do so because their configuration file has grown too large for their NVRAM to handle. It can also be a useful way of keeping an access list that is shared by a number of routers centralized and up-to-date. We have sometimes used it as a temporary measure when the NVRAM in a router is damaged.

However, we consider this feature to be highly risky and recommend avoiding it in most cases. If the problem is simply one of NVRAM capacity, Recipe 1.4 shows how to compress the startup configuration file to help fit more information into your existing NVRAM. Also, since routers can operate for years without reloading, using this feature to keep your routers up-to-date seems pointless.

If you choose to implement remote configuration despite these cautions, you need to understand how the boot process works. When you enable the *service config* option, the router attempts to load a

network file, then a host file. The router assumes that network files are common to all routers, while the host file contains router-specific information. If it can't find these files, the router will generate the following error messages:

```
%Error opening tftp://255.255.255.255/network-confg (Timed out)
%Error opening tftp://255.255.255.255/cisconet.cfg (Timed out)
%Error opening tftp://255.255.255.255/router1-confg (Timed out)
%Error opening tftp://255.255.255.255/router1.cfg (Timed out)
```

Here you can see what happened when we enabled the *service config* option and reloaded our router, which was called *router1*. It attempted to load several different files automatically. The first two files have generic network file names. The router then looks for the host file under two different names. It attempts to load these configuration files from IP address `255.255.255.255` by default.

When we add the *boot* commands, the router looks for the specified files from the appropriate TFTP server. Again, notice the order that the router loaded the files: the network file first, followed by the host file.

```
Loading Network-auto from 172.25.1.1 (via Ethernet0): !
[OK - 27/4096 bytes]

Loading Router1-auto from 172.25.1.1 (via Ethernet0): !
[OK - 71/4096 bytes]
```

If you do not configure the router to load specific network or host filenames, it will try to load the default files, shown in the trace above. If these files don't exist, the router will pause for a significant amount of time while it tries to find them. When you use this feature, you should always include both a network and a host file to load. If you don't need a network file, you can put a file on the server that contains only the keyword end.

> This feature loads configuration commands only into the running configuration. It does not copy them into the startup configuration file.

The *show version* tells you whether the router was able to load these files successfully:

```
Router1#show version
Cisco Internetwork Operating System Software
IOS (tm) 2500 Software (C2500-IO-L), Version 12.2(7a), RELEASE SOFTWARE (fc2)
Copyright (c) 1986-2002 by cisco Systems, Inc.
Compiled Thu 21-Feb-02 02:07 by pwade
Image text-base: 0x0304CF80, data-base: 0x00001000

ROM: System Bootstrap, Version 5.2(8a), RELEASE SOFTWARE
BOOTLDR: 3000 Bootstrap Software (IGS-RXBOOT), Version 10.2(8a), RELEASE SOFTWARE
(fc1)

Router1 uptime is 4 minutes
```

```
System returned to ROM by reload
System image file is "flash:c2500-io-l.122-7a.bin"
Host configuration file is "tftp://172.25.1.1/Router1-auto"
Network configuration file is "tftp://172.25.1.1/Network-auto"

cisco 2500 (68030) processor (revision D) with 16384K/2048K bytes of memory.
Processor board ID 04915359, with hardware revision 00000000
Bridging software.
X.25 software, Version 3.0.0.
2 Ethernet/IEEE 802.3 interface(s)
2 Serial network interface(s)
32K bytes of non-volatile configuration memory.
16384K bytes of processor board System flash (Read ONLY)

Configuration register is 0x2102
```

The *service config* option is disabled by default. However, if the router tries to boot but cannot find its startup configuration file, it will automatically enable this option and attempt to find a configuration file through the network:

```
00:00:25: AUTOINSTALL: Ethernet0 is assigned 172.25.1.30
00:00:25: AUTOINSTALL: Obtain siaddr 172.25.1.3 (as config server)
00:00:25: AUTOINSTALL: Obtain default router (opt 3) 172.25.1.3
%Error opening tftp://172.25.1.3/network-confg (No such file or directory)
%Error opening tftp://172.25.1.3/cisconet.cfg (No such file or directory)
%Error opening tftp://172.25.1.3/router-confg (No such file or directory)
%Error opening tftp://172.25.1.3/ciscortr.cfg (No such file or directory)
%Error opening tftp://172.25.1.3/network-confg (No such file or directory)
%Error opening tftp://172.25.1.3/cisconet.cfg (No such file or directory)
```

Two interesting things happen if you reload a router with an empty configuration file. First, the router enables its *autoinstall* option and attempts to acquire an IP address via DHCP. In this example, the router obtained a DHCP address of `172.25.1.30`. Second, after it obtains a dynamic address, it attempts to load a configuration file via TFTP.

Notice the filenames that the router cycles through in an attempt to load a configuration file. If there happens to be a file with one of these names in the TFTP directory, the router will download it and use its contents to configure itself. This can cause serious problems.

1.3.4 See Also

Recipe 1.4; Recipe 1.5

Top

Recipe 1.4 Storing Configuration Files Larger than NVRAM

1.4.1 Problem

Your configuration file has become larger than the router's available NVRAM.

1.4.2 Solution

You can compress your router's configuration file before saving it to NVRAM to allow you to save more configuration information. The command *service compress-config* will compress the configuration information when the router saves the file, and uncompress it when it is required:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#service compress-config
Router1(config)#end
Router1#
```

1.4.3 Discussion

Cisco generally ships its routers with more than enough NVRAM to store an average configuration file. However, there are times when configuration files exceed the available NVRAM. For instance, some routers contain large access lists that are hundreds of lines in length. When configuration files grow beyond the finite amount of NVRAM you will begin to have problems.

The first sign of serious problems with an overly large configuration file is usually when the router refuses to save its configuration because of size. This is a dangerous situation because the router can no longer keep a copy of the whole running-configuration file in its NVRAM storage, and it is difficult to predict how much of your configuration will be lost if you were to reload the router.

Turning on compression roughly doubles the size of the configuration file you can store. You must put the command *service compress-config* into the configuration with a configure terminal. Then, for this command to take effect, you need to copy the running configuration file to NVRAM:

```
Router1#copy running-config startup-config
Destination filename [startup-config]? <enter>
Building configuration...
Compressed configuration from 9664 bytes to 4903 bytes[OK]
Router1#
```

In this case, you can see that the compression reduced the configuration file to less than half of its original size. This compression algorithm will not attempt to compress a file that is three times larger than the

available NVRAM space. Although this limit exists, we have never seen a router approach a 3 to 1 ratio in practice.

The actual amount of available NVRAM storage varies between different router models. You can see how much total NVRAM storage is available on a particular router with the *show version* command:

```
Router1#show version
Cisco Internetwork Operating System Software
IOS (tm) C2600 Software (C2600-IK9O3S-M), Version 12.2(12a), RELEASE SOFTWARE (fc1)
Copyright (c) 1986-2002 by cisco Systems, Inc.
Compiled Tue 24-Sep-02 02:05 by pwade
Image text-base: 0x8000808C, data-base: 0x8127FF40

ROM: System Bootstrap, Version 11.3(2)XA4, RELEASE SOFTWARE (fc1)

Router1 uptime is 12 hours, 15 minutes
System returned to ROM by reload
System restarted at 23:18:45 EST Fri Jan 10 2003
System image file is "flash:c2600-ik9o3s-mz.122-12a.bin"

cisco 2621 (MPC860) processor (revision 0x102) with 45056K/4096K bytes of memory.
Processor board ID JAB04130B2Q (1293133440)
M860 processor: part number 0, mask 49
Bridging software.
X.25 software, Version 3.0.0.
2 FastEthernet/IEEE 802.3 interface(s)
2 Serial network interface(s)
32K bytes of non-volatile configuration memory.
16384K bytes of processor board System flash (Read/Write)

Configuration register is 0x2102

Router1#
```

This router contains 32Kb of NVRAM in which to store configuration files. The top of the output from the *show startup-config* command shows how much NVRAM storage is available, and how much this particular configuration file requires. If you enable compression, it will also show the compressed and uncompressed sizes:

```
Router1#show startup-config
Using 5068 out of 29688 bytes, uncompressed size = 9969 bytes
Uncompressed configuration from 5068 bytes to 9969 bytes
!
! Last configuration change at 12:36:22 EST Sat Jan 11 2003 by ijbrown
! NVRAM config last updated at 13:34:57 EST Sat Jan 11 2003 by ijbrown
!
version 12.2
<removed for brevity>
```

In this case we have used about 5Kb of the available 29Kb for this router's configuration file. However,

the *show version* output indicates a total of 32Kb NVRAM, leaving 3Kb unaccounted for. The router's NVRAM used to contain the startup configuration file only, but this is no longer strictly the case. Recent IOS releases also use the same NVRAM space to store information such as private keys for SSH or IPSec, and interface numbers for SNMP. You can see information about all of these files with the *dir nvram:* command:

```
Router1#dir nvram:
Directory of nvram:/

   20  -rw-        5068                <no date>  startup-config
   21  ----        2302                <no date>  private-config
    1  ----           0                <no date>  persistent-data
    2  -rw-         133                <no date>  ifIndex-table

29688 bytes total (20218 bytes free)
Router1#
```

Note that the second column from the left in this output contains file attributes similar to those used by the Unix *ls* command. In this case, both the *startup-config* and *ifIndex-table* files are readable and writable. For example, you could look at your router's *startup-config* file using the following command:

```
Router1#more nvram:/startup-config
```

You can view any file that has an "r", and you can modify any file that has a "w". The two files in this router's NVRAM that have neither "r" nor "w" can't be displayed, modified, or deleted. Note that although the *ifIndex-table* is readable, it is in a binary format that isn't very meaningful when displayed with the *more* command.

1.4.4 See Also

Recipe 1.3

Top

Recipe 1.5 Clearing the Startup Configuration

1.5.1 Problem

You want to clear an old configuration out of your router and return it to a factory default configuration.

1.5.2 Solution

You can delete the current startup configuration files and return the router to its factory default settings with the *erase nvram:* command:

```
Router1#erase nvram:
Erasing the nvram filesystem will remove all files! Continue? [confirm] <enter>
[OK]
Erase of nvram: complete
Router1#reload

System configuration has been modified. Save? [yes/no]: no
Proceed with reload? [confirm] <enter>
```

You can achieve the same result with the *erase startup-config* command:

```
Router1#erase startup-config
Erasing the nvram filesystem will remove all files! Continue? [confirm] <enter>
[OK]
Erase of nvram: complete
Router1#reload
Proceed with reload? [confirm] <enter>
```

1.5.3 Discussion

Before you redeploy an old router that you have previously used for another purpose, it is a good idea to completely erase the old configuration. This ensures that the router starts with a clean configuration. However, if you did this on a production router, it would wipe out the configuration and disable all of the interfaces. Fortunately, completely deleting your configuration requires two steps: you must erase the startup configuration, then reload the router.

After you erase your startup configuration file and reload, the router will enter its configuration dialog mode. Most experienced Cisco engineers prefer to skip this mode:

```
--- System Configuration Dialog ---

Would you like to enter the initial configuration dialog? [yes/no]: no

Would you like to terminate autoinstall? [yes]: yes

Press RETURN to get started!
Router>
```

At this point, the router's configuration has been returned to the factory defaults:

```
Router#show running-config
Building configuration...

Current configuration : 431 bytes
!
version 12.2
service timestamps debug uptime
service timestamps log uptime
no service password-encryption
!
hostname Router
!
!
ip subnet-zero
!
!
!
!
interface Ethernet0
 no ip address
 shutdown
!
interface Ethernet1
 no ip address
 shutdown
!
interface Serial0
 no ip address
 shutdown
!
interface Serial1
 no ip address
 shutdown
!
ip classless
ip http server
ip pim bidir-enable
!
!
line con 0
line aux 0
```

```
line vty 0 4
!
end

Router#
```

You can now safely reconfigure the router for its new function. Note that the factory defaults vary, depending on the level of IOS you are running and the hardware installed in the router.

If you accidentally erase the startup configuration file, you can still recover it if the router has not yet been reloaded. Simply copy the running configuration back to the startup configuration:

```
Router1#show startup-config
startup-config is not present
Router1#copy running-config startup-config
Building configuration...
[OK]
Router1#show startup-config
version 12.2
service timestamps debug datetime msec
service timestamps log datetime localtime
service password-encryption
!
hostname  Router1
<removed for brevity>
```

If the router's configuration is erased and the router is reloaded, it will either need to be reconfigured manually from memory, or preferably, from a backup copy (as in Recipe 1.2).

1.5.4 See Also

Recipe 1.2

Top

Recipe 1.6 Loading a New IOS Image

1.6.1 Problem

You want to upgrade the IOS image that your router uses.

1.6.2 Solution

The *copy tftp* command allows you to use TFTP to download a new IOS version into the router's Flash memory:

```
Router1#copy tftp://172.25.1.1/c2600-ik9o3s-mz.122-12a.bin  flash:
Destination filename [c2600-ik9o3s-mz.122-12a.bin]? <enter>
Accessing  tftp://172.25.1.1/c2600-ik9o3s-mz.122-12a.bin...
Erase flash: before copying? [confirm] <enter>
Erasing the flash filesystem will remove all files! Continue? [confirm] <enter>
Erasing
device... eeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeee ...
erased
Erase of flash: complete
Loading c2600-ik9o3s-mz.122-12a.bin from 172.25.1.1 (via FastEthernet0/0.1):
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
[OK - 11135588 bytes]

Verifying checksum...  OK (0xE643)
11135588 bytes copied in 82.236 secs (135410 bytes/sec)
Router1# reload
Proceed with reload? [confirm] <enter>
```

1.6.3 Discussion

Sooner or later you will need to upgrade your router's IOS image. Common reasons for upgrading the IOS include new features, bug fixes, and security vulnerabilities. Before you attempt to upgrade your IOS, you should save a backup copy of your current IOS to your TFTP server, as discussed in Recipe 1.9 .

You should always start by analyzing how much free space is available in your router's flash to ensure that there is enough room to load the new IOS image. If there isn't enough, then you may have to erase existing image(s) from flash—as we did in our example. In some cases, you may not have enough flash to load the new image at all. You can use the *show flash* command to see how much flash memory is

available:

```
Router1#show flash

System flash directory:
File  Length   Name/status
  1   11135588  c2600-ik9o3s-mz.122-12a.bin
[11135652 bytes used, 5117276 available, 16252928 total]
16384K bytes of processor board System flash (Read/Write)

Router1#
```

Some router models can support additional flash memory in the form of PCMCIA cards. The *show slot0:* and *show slot1:* commands will give you details about these additional storage locations. You can store IOS images on the flash cards, but keep in mind that the router will load the first available IOS image in the router's main flash by default. Recipe 1.7 shows how to boot the router using IOS images located on flash cards.

Before you can load an IOS image to your router, you must download the appropriate image from Cisco or purchase it on CD media. You will then need to move the new image into the TFTP directory and ensure that the file is world readable . On Unix systems, you can use the *chmod* command to do this. If the file is not world readable, the TFTP process will not be able to access it, and the IOS upgrade will fail. You should also do some simple sanity checks on the file by confirming the file size is correct and that the checksum or MD5 "fingerprint" match the values provided by Cisco. Note that it is extremely unwise to use an IOS image that is not created or supported by Cisco.

The router will do several tests while loading a new IOS image to ensure that the process goes smoothly. The first test is to see whether the TFTP server has the specified file:

```
Router1#copy tftp://172.25.1.1/c2600-ik9o3s-mz.122-14.bin  flash:
Destination filename [c2600-ik9o3s-mz.122-14.bin]?
Accessing  tftp://172.25.1.1/c2600-ik9o3s-mz.122-14.bin...
%Error opening tftp://172.25.1.1/c2600-ik9o3s-mz.122-14.bin (No such file or directory
Router1#
```

Here you can see that the router tried to find the file on the TFTP server before going any further, and discovered that it was not present. If the file exists and the permissions are correct, the router will continue with the dialogue. If your TFTP server keeps detailed logs of activity you will see an aborted TFTP file transfer as the router checked to see if the file was available. These aborted attempts are normal and shouldn't cause concern. If the requested file is present but not world readable, the router will show an error message and abort the upgrade process:

```
Router1#copy tftp://172.25.1.1/c2600-ik9o3s-mz.122-12a.bin  flash:
Destination filename [c2600-ik9o3s-mz.122-12a.bin]? <enter>
Accessing  tftp://172.25.1.1/c2600-ik9o3s-mz.122-12a.bin...
%Error opening tftp://172.25.1.1/c2600-ik9o3s-mz.122-12a.bin (Permission denied)
Router1#
```

Aborting the upgrade early in the process like this ensures that you don't erase the flash unless there is a suitable replacement image available for download.

In the next step of the download process, you must tell the router whether to erase the flash before downloading a new image. If there is enough room available in flash, you can load the new image without erasing the existing image or images. However, if you attempt to download an image and you don't have enough flash space available, the router will protest and abort the procedure:

```
Router1#copy tftp://172.25.1.1/c2600-ik9o3s-mz.122-12a.bin  flash:
Destination filename [c2600-ik9o3s-mz.122-12a.bin]? <enter>
Accessing  tftp://172.25.1.1/c2600-ik9o3s-mz.122-12a.bin...
Erase flash: before copying? [confirm]n
Loading c2600-ik9o3s-mz.122-12a.bin from 172.25.1.1 (via FastEthernet0/0.1): !
%Error copying tftp://172.25.1.1/c2600-ik9o3s-mz.122-12a.bin (Not enough space on dev
Router1#
```

The process of upgrading your router's IOS image is fairly forgiving. The router performs sanity checks throughout the process to ensure that image integrity is maintained. After downloading the image, the router verifies the image's checksum. This ensures that the IOS image was not corrupted during transmission. If the image does not pass the verification test, attempt your download again without reloading the router.

You can manually check to see if the IOS checksum is correct by using the *verify* command. We show how to use the *verify* command in Recipe 1.10.

If the IOS upgrade goes smoothly and the checksum verifies correctly, then it is safe to reboot your router and load the new IOS image. Once the router becomes reachable again, you should verify that the new IOS loaded correctly with the *show version* command:

```
Router1#show version
Cisco Internetwork Operating System Software
IOS (tm) C2600 Software (C2600-IK9O3S-M), Version 12.2(12a), RELEASE SOFTWARE (fc1)
Copyright (c) 1986-2002 by cisco Systems, Inc.
Compiled Tue 24-Sep-02 02:05 by pwade
Image text-base: 0x8000808C, data-base: 0x8127FF40

ROM: System Bootstrap, Version 11.3(2)XA4, RELEASE SOFTWARE (fc1)

Router1 uptime is 2 minutes
System returned to ROM by reload
System restarted at 11:53:26 EST Sat Jan 11 2003
System image file is "flash:c2600-ik9o3s-mz.122-12a.bin"

cisco 2621 (MPC860) processor (revision 0x102) with 45056K/4096K bytes of memory.
Processor board ID JAB04130B2Q (1293133440)
M860 processor: part number 0, mask 49
Bridging software.
X.25 software, Version 3.0.0.
2 FastEthernet/IEEE 802.3 interface(s)
```

```
2 Serial network interface(s)
32K bytes of non-volatile configuration memory.
16384K bytes of processor board System flash (Read/Write)

Configuration register is 0x2102

Router1#
```

In this example, you can see that the router's boot sequence completed successfully and that it's running the new IOS version. Also, notice that the router's new system image file matches the name of the file we just downloaded. This indicates the IOS upgrade was completely successful.

1.6.4 See Also

Recipe 1.7 ; Recipe 1.9 ; Recipe 1.10

Top

Recipe 1.7 Booting a Different IOS Image

1.7.1 Problem

You want to boot using an alternate IOS image.

1.7.2 Solution

To specify which IOS image the router should load next time it reboots, use the *boot system* command:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#boot system flash:c3620-jk9o3s-mz.122-7a.bin
Router1(config)#boot system flash:c3620-jos56i-l.120-11.bin
Router1(config)#boot system slot0:c3620-ik9s-mz.122-13.bin
Router1(config)#boot system rom
Router1(config)#end
```

> The sequence of the boot system commands is extremely important, as the router will attempt to load the IOS images in the order that they appear in the configuration file.

1.7.3 Discussion

The router can store as many IOS images in its flash memory as there is space to hold. If there is only one file, it can safely assume that this must be the IOS image to load. However, if the router has several images in its flash storage, you need to specify which one it should load, or the router will simply select one. This is particularly true on routers that have additional flash memory in the form of PCMCIA cards, which can hold many files—not all of them are necessarily IOS images.

With the default configuration register settings, the router will attempt to load the first accessible IOS image it finds in its flash storage. However, loading the first available image might not be appropriate. For instance, in our last recipe we showed that if you have space, you can download a new IOS image without erasing old images. In this case, you probably want the router to load the newer IOS image. It would be better still if the router tried the new image first, then reverted to the old image if the new one failed to load correctly for any reason. The *boot system* command allows you to specify not only which IOS images to boot from, but also the order in which to try them if the router has trouble booting.

In the example, this router tries a succession of three different IOS images. If they all fail, it resorts to using its boot ROM image.

As we noted earlier, the sequence of the boot system commands is important since the router will attempt to load the IOS images in order of entry. This means that the only way to add a new IOS image to the start of the list is to remove all of the old *boot system* commands and reenter them again in the order of preference. You can remove all of the boot system commands at once with the following command:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#no boot system
Router1(config)#end
Router1#
```

Once the old *boot system* commands have been removed, you can configure a new set in whatever order you require.

In addition to allowing you to boot from IOS images in the router's flash storage, you can also use the *boot system* command to boot from the IOS image in its ROM storage, or by using the TFTP or remote copy (RCP) protocols across the network. Recipe 1.8 shows an example of booting across the network using TFTP. Table 1-1 shows the options for the *boot system* command.

Table 1-1. Boot system command target options

Keyword	Description
flash:	On-board flash
slot0:	PCMCIA flash card in slot0
slot1:	PCMCIA flash card in slot1
mop	Load an image using the MOP protocol
bootflash:	Load bootflash image (not available on all systems)
rom:	Load the image from ROM
rcp:	Load an image using the RCP
tftp:	Load an image using the TFTP protocol

In addition to the *boot system* commands, you can use the router's configuration register to change which image the router boots from. The last octet in the configuration register must be set to "2" or the router will completely ignore the *boot system* commands. For instance, if the last octet of the configuration

register is set to "1", the router will boot from ROM and ignore the *boot system* commands. In our example, the test router's configuration register was set to `0x2102`. The *config-register* command allows you to set the appropriate configuration register values:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#config-register 0x2102
Router1(config)#end
Router1#
```

It is important to remember that, unlike any other configuration command, you don't need to save the running configuration to NVRAM when you change the configuration register setting. It will survive a reload without being saved. In fact, the new setting will not take effect until after the next reload:

```
Router1#show version
Cisco Internetwork Operating System Software
IOS (tm) 3000 Bootstrap Software (IGS-RXBOOT), Version 10.2(8a), RELEASE SOFTWAR
E (fc1)
Copyright (c) 1986-1995 by cisco Systems, Inc.
Compiled Tue 24-Oct-95 15:46 by mkamson
Image text-base: 0x01020000, data-base: 0x00001000

ROM: System Bootstrap, Version 5.2(8a), RELEASE SOFTWARE

Router1 uptime is 2 minutes
System restarted by reload
Running default software

cisco 2500 (68030) processor (revision D) with 16380K/2048K bytes of memory.
Processor board serial number 04915359 with hardware revision 00000000
X.25 software, Version 2.0, NET2, BFE and GOSIP compliant.
2 Ethernet/IEEE 802.3 interfaces.
2 Serial network interfaces.
32K bytes of non-volatile configuration memory.
16384K bytes of processor board System flash (Read/Write)

Configuration register is 0x2101 (will be 0x2102 at next reload)

Router1#
```

After setting the appropriate *boot system* commands and reloading the router, you can use the *show version* command to see which image file the router used to boot:

```
Router2#show version
Cisco Internetwork Operating System Software
IOS (tm) 3600 Software (C3620-IK9S-M), Version 12.2(13), RELEASE SOFTWARE (fc1)
Copyright (c) 1986-2002 by cisco Systems, Inc.
Compiled Tue 19-Nov-02 19:04 by pwade
Image text-base: 0x60008930, data-base: 0x61276000
```

```
ROM: System Bootstrap, Version 11.1(19)AA, EARLY DEPLOYMENT RELEASE SOFTWARE (fc1)

Router2 uptime is 2 hours, 4 minutes
System returned to ROM by reload
System restarted at 21:13:13 EST Wed Jan 15 2003
System image file is "slot0:c3620-ik9s-mz.122-13.bin"

cisco 3620 (R4700) processor (revision 0x81) with 41984K/7168K bytes of memory.
Processor board ID 05969532
R4700 CPU at 80Mhz, Implementation 33, Rev 1.0
Bridging software.
X.25 software, Version 3.0.0.
SuperLAT software (copyright 1990 by Meridian Technology Corp).
Basic Rate ISDN software, Version 1.1.
1 Ethernet/IEEE 802.3 interface(s)
1 FastEthernet/IEEE 802.3 interface(s)
1 Serial network interface(s)
1 ISDN Basic Rate interface(s)
DRAM configuration is 32 bits wide with parity disabled.
29K bytes of non-volatile configuration memory.
16384K bytes of processor board System flash (Read/Write)
16384K bytes of processor board PCMCIA Slot0 flash (Read/Write)
16384K bytes of processor board PCMCIA Slot1 flash (Read/Write)

Configuration register is 0x2102

Router2#
```

In this case, the router says that it loaded its IOS image from *slot0:*, as configured. After changing your *boot system* commands, you should make sure to reboot and verify that the router behaves as expected. You don't want the wrong IOS image to accidentally get loaded the next time the router reboots. If you do have problems with the *boot system* command, connect to the console and reload the router. This will display any error messages as the router boots. The router does not capture these messages anywhere; this is the only way to see them.

1.7.4 See Also

Recipe 1.6; Recipe 1.8

Top

Recipe 1.8 Booting Over the Network

1.8.1 Problem

You want to load an IOS image that is too large to store on your router's local flash.

1.8.2 Solution

You can load an IOS image that is larger than your router's flash by configuring the router to use TFTP to download the image before booting:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#boot system tftp c2500-io-l.122-7a.bin 172.25.1.1
Router1(config)#boot system flash
Router1(config)#end
Router1#
```

1.8.3 Discussion

We mentioned in Recipe 1.7 that it is possible to load IOS images over the network at boot time. However, booting from remote IOS images presents some unique challenges. This is why we have dedicated an entire recipe to remote booting.

One of the most important advantages of booting an IOS image over the network is that it allows you to use images that are larger than your router's flash. Like any other software, each new IOS image tends to be slightly larger than the previous versions. It is relatively common to discover that you can't load the latest IOS version because it is too big to fit in an older router's flash.

Booting over the network also provides a way of loading a backup IOS image if the primary image fails. As we discussed Recipe 1.7, it's a good idea to configure your router with at least one backup IOS image to load in case the primary fails for any reason. Even if you have a lot of flash storage, you may find that you can't store two IOS images at once. So booting over the network is actually a reasonable way of providing a backup image.

Booting over the network also poses an important security problem because, as we discussed in Recipe 1.2, it's virtually impossible to secure a UDP-based service like TFTP. In addition, it makes the router dependent on the TFTP server for its boot images. Network booting also has performance issues. Loading an IOS over the network can significantly increase the time it takes your router to reload,

particularly if it has to traverse slower WAN links. We certainly do not recommend relying solely on remote booting in a production environment.

However, in a lab or testing environment, the ability to load an IOS image that is larger than your router's flash can be extremely useful. Booting over the network lets you work with IOS versions that you could not otherwise load and test. The following *show version* command output is from a router that was booted this way:

```
Router1#show version
Cisco Internetwork Operating System Software
IOS (tm) 2500 Software (C2500-IO-L), Version 12.2(7a), RELEASE SOFTWARE (fc2)
Copyright (c) 1986-2002 by cisco Systems, Inc.
Compiled Thu 21-Feb-02 02:07 by pwade
Image text-base: 0x0000144C, data-base: 0x0082E874
ROM: System Bootstrap, Version 5.2(8a), RELEASE SOFTWARE
BOOTLDR: 3000 Bootstrap Software (IGS-RXBOOT), Version 10.2(8a), RELEASE SOFTWARE
(fc1)
Router1 uptime is 10 hours, 16 minutes
System returned to ROM by reload
System restarted at 01:57:47 EST Sat Jan 11 2003
System image file is "tftp://172.25.1.1/c2500-io-l.122-7a.bin"
cisco 2520 (68030) processor (revision E) with 16384K/2048K bytes of memory.
Processor board ID 03870281, with hardware revision 00000002
Bridging software.
X.25 software, Version 3.0.0.
Basic Rate ISDN software, Version 1.1.
1 Ethernet/IEEE 802.3 interface(s)
2 Serial network interface(s)
2 Low-speed serial(sync/async) network interface(s)
1 ISDN Basic Rate interface(s)
32K bytes of non-volatile configuration memory.
16384K bytes of processor board System flash (Read/Write)
Configuration register is 0x2102
Router1#
```

This shows that the router is running the new version of IOS, which it loaded using TFTP. In this example, we put the TFTP boot first:

```
Router1(config)#boot system tftp c2500-io-l.122-7a.bin 172.25.1.1
Router1(config)#boot system flash
```

If the TFTP file transfer had failed, the router would have loaded its old IOS image from its local flash. If we had reversed the order of these commands, the router would have tried first to boot from flash, then resorted to TFTP if it had trouble with the file on the flash.

For redundancy purposes, you can configure the router to boot from multiple TFTP servers. Simply copy the same IOS image to an alternate set of TFTP servers and include a *boot system* command for each server. This reduces the dependency of the router to a single TFTP server, but the router has to try each successive server and time out before moving on to the next one. This can increase the boot time.

1.8.4 See Also

Recipe 1.2; Recipe 1.7

Top

◀ Previous Next ▶

Recipe 1.9 Copying an IOS Image to a Server

1.9.1 Problem

You want to save a backup copy of your IOS image on a TFTP server.

1.9.2 Solution

You can upload a copy of your router's IOS image to a TFTP server with the following set of commands:

```
Freebsd% touch /tftpboot/c2600-ik9o3s-mz.122-12a.bin
Freebsd% chmod 666 /tftpboot/c2600-ik9o3s-mz.122-12a.bin
Freebsd% telnet Router1
Trying 172.25.1.5...
Connected to Router1.
Escape character is '^]'.

User Access Verification

Password: <vtypassword>

Router1>en
Password: <enablepassword>
Router1#copy flash:c2600-ik9o3s-mz.122-12a.bin   tftp
Address or name of remote host [  ]? 172.25.1.1
Destination filename [c2600-ik9o3s-mz.122-12a.bin]? <enter>
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
11135588 bytes copied in 52.588 secs (211752 bytes/sec)
Router1#
```

1.9.3 Discussion

It's a good idea to save a copy of the current IOS image before attempting to upgrade the IOS version of a router. This way, if an upgrade fails or if you have problems with the new IOS version, you can revert back to the old proven IOS version. The procedure to copy an IOS image to a TFTP server is very similar to the way we backed up a configuration file in Recipe 1.2 The only real difference is the size of the file involved—IOS images are quite a bit larger than configuration files.

As we mentioned in Recipe 1.2, you have to verify the file permissions on your TFTP server. The

transfer will fail if this file isn't world writable. We highly recommend that you remove the world writable attribute on this file after uploading it. On Unix systems, you can use the *chmod* command to change the file attributes. This will ensure that the file isn't accidentally overwritten. Unlike configuration files (which you should never store in your TFTP directory), *world writeable* IOS images pose no security concerns.

1.9.4 See Also

Recipe 1.2 ; Recipe 1.6

Top

Recipe 1.10 Copying an IOS Image Through the Console

1.10.1 Problem

You want to load an IOS image into your router through a serial connection to the console or AUX ports.

1.10.2 Solution

You can use the following set of commands to copy an IOS image onto a router through the console or the AUX port:

```
Router1#copy xmodem: slot1:
                          **** WARNING ****
x/ymodem is a slow transfer protocol limited to the current speed
settings of the auxiliary/console ports. The use of the auxilary
port for this download is strongly recommended.
During the course of the download no exec input/output will be
available.
                          ---- ******* ----

Proceed? [confirm] <enter>
Destination filename [  ]? c3620-ik9s-mz.122-12a.bin
Erase slot1: before copying? [confirm] <enter>
Use crc block checksumming? [confirm] <enter>
Max Retry Count [10]: <enter>
Perform image validation checks? [confirm] <enter>
Xmodem download using crc checksumming with image validation
Continue? [confirm]  <enter>
Ready to receive file..........CC  <start xmodem file transfer here>
4294967295 bytes copied in 1450.848 secs (1271445669961 bytes/sec)
Router1#
```

> Cisco highly recommends using the AUX port for this procedure rather than the console port because the AUX port supports hardware flow control.

1.10.3 Discussion

It can be quite useful to be able to load an IOS image through a serial connection, particularly if you don't have access to a TFTP server, or if the router doesn't have any accessible LAN interfaces.

Although this feature is rarely used, Cisco supports *xmodem* and *ymodem* file transfers through a serial connection.

We also recommend enabling the CRC checksum feature when you use *xmodem* to download an IOS image through a serial connection. This will help to ensure the integrity of the file transfer.

We should stress that this process can be extremely slow. Don't even attempt to download an IOS image at the default speed of 9600bps unless you have an entire day to kill. We highly recommend increasing the speed to the highest value that your terminal emulation package will support. We have found that 115200bps provides the maximum throughput with the greatest reliability. The *speed* command allows you to change the speed of an asynchronous serial port:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#line aux 0
Router1(config-line)#speed 115200
Router1(config-line)#end
Router1#
```

In this example, we used *Hyperterminal* because it is included with the Windows operating system. However, almost any terminal emulation program that supports *xmodem* or *ymodem* protocols will work. In fact, we have found significant differences in download times between the various emulation packages, and HyperTerminal tends to be one of the slowest. Other packages such as ProComm tend to be somewhat faster. But they all work.

Even after we increased the speed of the AUX port to 115200bps the file transfer took nearly 25 minutes to complete. By comparison, loading the same IOS version via TFTP through an Ethernet connection took less than 4 minutes. So, in general, we don't recommend using this method unless you can't use TFTP.

The first step, once you have a copy of the IOS image on your computer, is to connect to the router's AUX port. Set the line speed to 115200bps on both the router's port and the terminal emulator, and issue the *copy* command. The router will prompt you to begin the file transfer with the text "Ready to receive file."

At this point, you should begin your file transfer protocol. If you are using HyperTerminal, select the "Transfer" drop-down menu, and then click on "Send-file." It will prompt you for filename and location and protocol type. Enter the name of the IOS image, then select "Xmodem" to start the file transfer.

During the file transfer, the connection is busy transferring the file, so the router can't display any messages. This is normal. However, most terminal emulator programs provide a status window to let you keep track of the file transfer.

When the transfer is complete, the terminal emulator drops out of the file transfer mode and the router puts up its normal prompt. At this point, we highly recommend checking the new IOS image to make

sure that it copied successfully. You can verify the file size as follows:

```
Router1#show slot1:

PCMCIA Slot1 flash directory:
File  Length   Name/status
  1   11922512  c3620-ik9s-mz.122-12a.bin
[11922576 bytes used, 4592496 available, 16515072 total]
16384K bytes of processor board PCMCIA Slot1 flash (Read/Write)
```

In this case, we loaded the image into the PCMCIA device in slot 1. If you put the image somewhere else, such as the internal flash memory, you would use the command *show flash:* instead.

If the file size is correct, you should check the image's checksum using the *verify* command:

```
Router1#verify slot1:c3620-ik9s-mz.122-12a.bin
Verified  slot1:c3620-ik9s-mz.122-12a.bin
Router1#
```

Top

Recipe 1.11 Deleting Files from Flash

1.11.1 Problem

You want to erase files from your router's flash.

1.11.2 Solution

To delete all of the files from your router's flash memory, use the *erase* command:

```
Router1#erase slot1:
Erasing the slot1 filesystem will remove all files! Continue? [confirm] <enter>
Erasing device...
eeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeee
eeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeee ...erased
Erase of slot1: complete
Router1#
```

Not all router types support the *erase* command.

You can remove individual files from the router's flash memory with the *delete* command:

```
Router1#delete slot1:c3620-ik9s-mz.122-13.bin
Delete filename [c3620-ik9s-mz.122-13.bin]? <enter>
Delete slot1:c3620-ik9s-mz.122-13.bin? [confirm] <enter>
Router1#
```

1.11.3 Discussion

As we have indicated, there are two ways to delete files from flash, depending on the type of router. The difference arose because Cisco routers use three different kinds of *filesystems*, called Class A, Class B, and Class C. Table 1-2 shows the filesystems that Cisco's most common routers use.

Table 1-2. Supported filesystems of common Cisco routers

Router type	Filesystem type
7000(RSP)	Class A
7500(RSP2,4, & 8)	Class A
12000	Class A
Route Switch Module (RSM)	Class A
1600	Class B
2500	Class B
3600[1]	Class B
4000	Class B
AS5300	Class B
AS5800	Class C
7100	Class C
7200	Class C

[1] The 3600 traditionally uses the Class B filesystem. However, starting with IOS Version 12.2(4)T, the 3600 also includes Class C filesystem functionality.

Table 1-3 lists some of the different filesystem commands, their meanings, and the filesystems that they work with.

Table 1-3. Filesystem commands

Command	Filesystem	Description
Delete	All	Marks the file as deleted, but does not permanently remove it from flash
Squeeze	A	Permanently removes all files that have been marked as deleted
Format	A & C	Erases the entire flash device
Verify	All	Verifies that the IOS file's checksum matches the value encoded in the image
Undelete	A & B	Recovers deleted files
Erase	A & B	Erases the entire flash device

The *erase* command is not available on all router types. On routers that use the Class C filesystem, you can only remove files from the flash with the *delete* command.

The *delete* command marks files as deleted, but it does not permanently remove them:

```
Router1#show slot1:

PCMCIA Slot1 flash directory:
File  Length   Name/status
  1   11992088  c3620-ik9s-mz.122-13.bin [deleted]
[16515072 bytes used, 0 available, 16515072 total]
16384K bytes of processor board PCMCIA Slot1 flash (Read/Write)

Router1#
```

You can permanently remove a file and reclaim the space on the flash device with the *squeeze* command. Note, however, that only routers with the type A filesystem support this command:

```
Router1#squeeze slot1:
Squeeze operation may take a while. Continue? [confirm] <enter>
squeeze in progress...
Squeeze of slot1 complete
Router1#
```

The squeeze function can take up to several minutes, so be patient. Once the *squeeze* command is complete, you can view the flash device to verify that the file is gone:

```
Router1#show slot1:

PCMCIA Slot1 flash directory:
No files in PCMCIA Slot1 flash
[0 bytes used, 16515072 available, 16515072 total]
16384K bytes of processor board PCMCIA Slot1 flash (Read/Write)

Router1#
```

The file has now been permanently removed and you can no longer recover it with the *undelete* command. On routers with filesystems that do not support the *squeeze* command, the only way to permanently remove deleted files is to use the *erase* command. However, the *erase* command deletes the entire flash system and will not permit you to delete individual files. In the next recipe, we look at ways to partition flash devices in order to reduce the impact of the *erase* command.

1.11.4 See Also

Recipe 1.12

Top

Recipe 1.12 Partitioning Flash

1.12.1 Problem

You want to change how your router's flash memory is partitioned.

1.12.2 Solution

The *partition* command allows you to create a partition in the router's flash memory:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#partition slot1: 2 8 8
Router1(config)#end
Router1#
```

1.12.3 Discussion

As we discussed in Recipe 1.11, the *erase* command deletes the entire contents of a flash device. On routers that don't support the *delete* and *squeeze* commands, there is no way to delete an individual file from flash without erasing all of the files in the flash device. Fortunately, you can use the *partition* command on flash devices to shelter some files from the effects of the *erase* command.

After you have partitioned a flash device, the *erase* command only affects one partition at a time. This command doesn't affect any of the other partitions on the same flash device. You can use this to allow you to delete individual files without having to wipe out the entire flash device.

In the next example, we partitioned a flash device into two equal parts. We then stored an IOS image on each of the partitions. You can see the partitions and their contents with the following command:

```
Router1#show slot1:

PCMCIA Slot1 flash directory, partition 1:
File  Length   Name/status
  1   7723664  c3620-ajs56i-mz.120-25.bin
[7723728 bytes used, 664880 available, 8388608 total]
8192K bytes of processor board PCMCIA Slot1 flash (Read/Write)

PCMCIA Slot1 flash directory, partition 2:
File  Length   Name/status
  1   7723664  c3620-ajs56i-mz.120-25.bin
```

```
[7723728 bytes used, 402736 available, 8126464 total]
8192K bytes of processor board PCMCIA Slot1 flash (Read/Write)

Router1#
```

Note that the router treats the two partitions as if they were separate flash devices. You can erase the contents of a particular partition by specifying the flash device name followed by the partition number and a colon:

```
Router1#erase slot1:2:
Erasing the slot1:2 filesystem will remove all files! Continue? [confirm] <enter>
Erasing device... eeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeee
...erased
Erase of slot1:2: complete
Router1#
```

If you view the entire flash device again, you can see that the file in partition 2 has been erased, while the contents of partition 1 remain untouched:

```
Router1#show slot1:

PCMCIA Slot1 flash directory, partition 1:
File  Length   Name/status
  1   7723664  c3620-ajs56i-mz.120-25.bin
[7723728 bytes used, 664880 available, 8388608 total]
8192K bytes of processor board PCMCIA Slot1 flash (Read/Write)
PCMCIA Slot1 flash directory, partition 2:
No files in PCMCIA Slot1 flash
[0 bytes used, 8126464 available, 8126464 total]
8192K bytes of processor board PCMCIA Slot1 flash (Read/Write)

Router1#
```

As we mentioned in Recipe 1.11, attempting to erase one file from this flash device without partitioning it first causes the router to erase both IOS images.

You can remove an existing set of partitions with the *no partition* command:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#no partition slot1:  2 8 8
ee
Router1(config)#end
Router1#
```

You can safely partition a flash device that already contains files as long as you don't attempt to create a partition partway through an existing file. If you attempt to create a partition that divides an existing file, the router will identify it as a problem.

1.12.4 See Also

Recipe 1.6; Recipe 1.11

Top

Recipe 1.13 Using the Router as a TFTP Server

1.13.1 Problem

You want to configure your router to act as a TFTP server.

1.13.2 Solution

The *tftp-server* command configures the router to act as a TFTP server:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#tftp-server flash:c2600-ik9o3s-mz.122-12a.bin
Router1(config)#end
Router1#
```

1.13.3 Discussion

The ability to use a router as a TFTP server can be quite useful. We have often used this feature to upgrade several routers that are separated from the TFTP server by slow WAN connections. In situations like this, you can upgrade one of the remote routers using TFTP over the slow WAN connection as we described in Recipe 1.6. Then you can configure this router to act as a TFTP server, and use it to upgrade the remaining routers over high-speed local links.

However, the router is not a fully functional TFTP server. It can only serve files for download. You cannot use this feature to upload files into the serving router's local flash. The router is not limited to just serving IOS images: you can use your router's flash to store configuration files and make them available for download via TFTP as well. You can even use it to hold configuration files for non-Cisco equipment.

Security is a concern whenever you enable services on a router. Every extra service you enable provides the wily hacker with a new potential avenue to exploit against your network. Therefore, we don't recommend using the TFTP server feature on routers facing the public Internet or other potentially unfriendly networks. However, for internal use, we believe it is reasonably safe. You can increase the security of the router's TFTP server by using an access list like this:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#access-list 99 permit 172.25.1.0 0.0.0.255
```

```
Router1(config)#access-list 99 deny any
Router1(config)#tftp-server flash:c2600-ik9o3s-mz.122-12a.bin 99
Router1(config)#end
Router1#
```

In this example, we defined an access list, 99, that will allow all devices on the `172.25.1.0/24` network to access the router's TFTP server. Then we applied the access list to the TFTP service by specifying the access list number at the end of the *tftp-server* command line. This will help to ensure that only the authorized devices permitted by the access list may download the specified file via TFTP.

You can configure the router to serve multiple files via TFTP by simply adding more *tftp-server* commands. If security is a concern, you can configure a different access list for each file.

Although this feature can be useful, we recommend enabling it only when you need to perform a download. Disabling the service as soon as the download has completed mitigates the security concerns of running extra services from your router.

1.13.4 See Also

Recipe 1.6

Top

Recipe 1.14 Using FTP from the Router

1.14.1 Problem

You want to use FTP directly from your router to download configuration or IOS files.

1.14.2 Solution

The *copy ftp:* command lets the router exchange files using FTP:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#ip ftp username ijbrown
Router1(config)#ip ftp password ianpassword
Router1(config)#end
Router1#copy ftp: running-config
Address or name of remote host [172.25.1.1]? 172.25.1.1
Source filename [  ]? test
Destination filename [running-config]? <enter>
Accessing  ftp://172.25.1.1/test...
Loading /test
[OK - 24/4096 bytes]

24 bytes copied in 0.276 secs (87 bytes/sec)
Router1#
```

We explicitly defined a username and password in this example. If you don't specify a username, the router will try to connect to the server's anonymous FTP service.

1.14.3 Discussion

Several recipes in this chapter show how to transfer files between your router and server using TFTP. However, Cisco routers also support FTP, which is better suited for transferring files over busy and congested links. While TFTP file transfers tend to abort if they encounter persistent congestion, FTP appears to be more resilient.

FTP is also somewhat more secure than TFTP because it uses usernames and passwords. TFTP has no user level security features. However, FTP sends its passwords across the network in unencrypted cleartext, so it is still not highly secure.

In the example we explicitly configured a FTP username and password on the router. Once this information is defined, using FTP is as easy as using TFTP. You can also override the username and password settings defined in the configuration file by including them on the command line:

```
Router1#copy ftp://ijbrown:ianpassword@172.25.1.1/c3620-ik9s-mz.122-10a.bin  slot1:
Destination filename [c3620-ik9s-mz.122-10a.bin]? <enter>
Accessing  ftp://ijbrown:ianpassword@172.25.1.1/c3620-ik9s-mz.122-10a.bin...
Loading pub/c3620-ik9s-mz.122-10a.bin !!!!
Erase slot1: before copying? [confirm] <enter>
Erasing the slot1 filesystem will remove all files! Continue? [confirm] <enter>
Erasing device... eeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeee ...erased
Erase of slot1: complete
Loading pub/c3620-ik9s-mz.122-10a.bin
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
[OK - 11819052/4096 bytes]

Verifying checksum...  OK (0x3238)
11819052 bytes copied in 266.956 secs (44273 bytes/sec)
Router1#
```

You can use URL format to specify the username, password, server address, and file you want to download. The format of the FTP URL looks like this:

    ftp://ijbrown:ianpassword@172.25.1.1/c3620-ik9s-mz.122-10a.bin

A colon separates the username, *ijbrown*, from the password, *ianpassword*. An @ sign then separates the user information from the server information, which can be either an IP address or a DNS name. A forward slash (*/*) separates the server name or address from the directory and filename.

If you don't include an FTP username in the configuration or the command line, the router will default to using anonymous FTP. If no password is specified in either the router's configuration or on the command line, the router will use a default password of mailto:router@cisco.com.

It is important to remember that if you specify a username and password on the command line, it will override whatever values you have configured. If you don't specify a username or password on the command line, the router will use the configured FTP username and password. And, if you don't specify username and password in either place, the router will resort to anonymous FTP.

1.14.4 See Also

Recipe 1.1; Recipe 1.2; Recipe 1.6; Recipe 1.10

Top

Recipe 1.15 Generating Large Numbers of Router Configurations

1.15.1 Problem

You need to generate hundreds of router configuration files for a big network rollout.

1.15.2 Solution

When building a large WAN, you will usually configure the remote branch routers similarly according to a template. This is a good basic design principle, but it also makes it relatively easy to create the router configuration files. Example 1-1 uses a Perl script to merge a CSV file containing basic router information with a standard template file. It takes the CSV file as input on STDIN.

Example 1-1. create-configs.pl

```
#!/usr/local/bin/perl
#
$template_file_name="rtr-template.txt";
while(<>) {

    ($location, $name, $lo0ip, $frameip, $framedlci, $eth0ip, $x)
        = split (/,/);

   open(TFILE, "< $template_file_name") || die "config template file $template_file_name
$!\n";
   $ofile_name = $name . ".txt";
   open(OFILE, "> $ofile_name") || die "output config file $ofile_name: $!\n";

   while (<TFILE>) {

     s/##location##/$location/;
     s/##rtrname##/$name/;
      s/##eth0-ip##/$eth0ip/;
      s/##loop0-ip##/$lo0ip/;
     s/##frame-ip##/$frameip/;
     s/##frame-DLCI##/$framedlci/;

     printf OFILE $_;
   }
}
```

1.15.3 Discussion

This Perl script is a simplified version of much longer scripts that we have used to create the

configuration files for some very large networks. After loading these configuration files into the routers, we shipped them to the remote locations along with a hard copy of the configuration in case there were problems during shipment. The technician installing the router could then simply connect the appropriate cables and power on the router. He wouldn't even need to log on to the router's console unless there were unexpected problems. This methodology can save hundreds of hours in a network installation project.

The script does a relatively simple merge function and expects the input data in CSV format on STDIN. For an input file named *RTR-DATA.CSV* , run the script as follows:

```
Freebsd% create-configs.pl < RTR-DATA.CSV
```

The input file in this case might look something like this:

```
Toronto, Router1, 172.25.15.1, 172.25.16.6, 101, 172.25.100.1,
Boston, Router2, 172.25.15.2, 172.25.16.10, 102, 172.25.101.1,
San Francisco, Router3, 172.25.15.3, 172.25.16.14, 103, 172.25.102.1,
```

Using a CSV file like is convenient because you can keep track of the entire network in a spreadsheet and create a CSV file containing the data you need for the router configurations.

The template configuration needs to include unique variable names that the script will replace with values from the CSV file. For example, the template configuration file might look like this:

```
!
version 12.1
service timestamps debug datetime msec
service timestamps log datetime msec
service password-encryption
!
hostname ##rtrname##
!
enable password cisco
enable secret cisco
!
interface  Loopback0
 ip address ##loop0-ip## 255.255.255.255
!
interface  Serial0/0
 description Frame-Relay Circuit
 no ip address
 encapsulation frame-relay
 ip route-cache policy
 frame-relay lmi-type ansi
 no shutdown
!
interface  Serial0/0.1 point-to-point
 ip address ##frame-ip## 255.255.255.252
 frame-relay interface-dlci ##frame-DLCI##
```

```
!
interface  FastEthernet0/1
 description User LAN Segment
 ip address ##eth0-ip## 255.255.255.0
 no shutdown
!
router eigrp 99
 network  172.25.0.0
!
snmp-server location ##location##
!
line con 0
 password cisco
 login
 transport input none
line aux 0
 password cisco
 login
line vty 0 4
 password cisco
 login
 transport input telnet
!
end
```

The script expects to find this template file located in the current directory with the name rtr-template.txt by default, but you can change this easily by modifying the variable called template_file_name in the script.

Naturally, your router templates will not look like this one, and your CSV file will almost certainly contain other data that is important to your network. This script will probably need significant local modification every time you use it on a new network, but the amount of time required to modify the script is usually far less than the amount of time needed to create all of the configuration files by hand.

The output of this script will be a series of files whose names are the same as the router names, but with *.txt* added to the end. You can then use a terminal emulator to cut and paste the router configuration files into the routers prior to shipping them to their destinations. Always remember to save the configuration to the router's NVRAM before powering it off:

```
Router1#copy running-config startup-config
```

1.15.4 See Also

Recipe 1.16 .

Top

Recipe 1.16 Changing the Configurations of Many Routers at Once

1.16.1 Problem

You want to make a configuration change to a large number of routers.

1.16.2 Solution

The Expect script in Example 1-2 makes the same configuration changes to a list of routers using Telnet. When it finishes running, the script produces a status report that identifies which devices, if any, failed to update properly. No arguments are required or expected.

Example 1-2. rtrchg.exp

```
#!/usr/local/bin/expect
#
#    rtrcfg.exp -- a script to perform mass configuration changes to
#                  a list of routers using Telnet and Expect
#
#
# Set Behavior
set tftp "172.25.1.1"
set workingdir /home/cisco/rtr
#
puts stdout "Enter user name:"
gets stdin userid
system stty -echo
puts stdout "Enter login password:"
gets stdin vtypasswd
puts stdout "\nEnter enable password:"
gets stdin enablepwd
system stty echo
system "cp $workingdir/NEWCONFIG /tftpboot/NEWCONFIG"
set RTR [open "$workingdir/RTR_LIST" r]
set LOG [open "$workingdir/RESULT" w]
while {[gets $RTR router] != -1} {
   if {[ string range $router 0 0 ] != "#"} {
     set timeout 10
     spawn telnet; expect "telnet>"; send "open $router\n"
     expect {
             {Username}   { send "$userid\r"
                            expect {
```

```
                                        {*Password*} { send "$vtypasswd\r" }
                                    }
                            }
                      {Password}    { send "$vtypasswd\r" }
                      timeout       { puts $LOG "$router - telnet failed"
                                      close; wait; continue
                                    }
                }

        expect {
                      {Password}    { puts $LOG "$router - vty login failed"
                                      close; wait; continue
                                    }
                      {Username}    { puts $LOG "$router - vty login failed"
                                      close; wait; continue
                                    }
                      {>}           { puts $LOG "$router - vty login ok" }

                       timeout      { puts $LOG "$router - vty login failed"
                                      close; wait; continue
                                    }
               }

        send "enable\r"
        expect "Password"
        send "$enablepwd\r"
        #
        expect {
                 {*#}           { puts $LOG "$router - enable login ok" }

                 {*>}           { puts $LOG "$router - enable login failed"
                                  close; wait; continue
                                }

                 timeout        { puts $LOG "$router - enable login failed"
                                  close; wait; continue
                                }
               }
# CMDs
set timeout 30
send "copy tftp://$tftp/NEWCONFIG running-config\r"
expect "running-config"
send "\r"
expect  {
          {OK}          { puts $LOG "$router - TFTP successful"}
          timeout       { puts $LOG "$router - TFTP failed"
                          close; wait; continue }
        }
send "copy running-config startup-config\r\r\r"
expect  {
          {OK}          { puts $LOG "$router - config saved"}
          timeout       { puts $LOG "$router - config failed"
```

```
                                close; wait; continue }
                }
      #CMDs
      send "exit\r"; close; wait
    }
}
close $RTR; close $LOG
system "rm /tftpboot/NEWCONFIG"
```

1.16.3 Discussion

This script uses the Expect language to emulate how a human router engineer might perform a series of configuration updates via TFTP. The script logs into each router in a list and uses TFTP to download a set of configuration changes into the router's running configuration. It then saves the new configuration file to NVRAM and moves on to the next router in the list. Automating a routine but time-consuming procedure with a script like this saves time and decreases the chances of fatigue-induced errors.

The script is designed to work with either normal router passwords or the AAA-enabled username and password combinations described in Chapter 3. The script begins by asking you to supply a username. If one or more of your routers aren't configured to use AAA or local authentication, the script simply ignores this username and inserts the password.

After asking for a username, the script prompts you to enter your VTY or AAA password. Then the script asks for the enable password and begins to perform the required configuration changes.

> The script is designed to work with IOS versions 12.0 and greater. Unfortunately, Cisco changed the command sequence prior to 12.0, meaning the script will not work correctly with old IOS versions.

You must change two variables in this script for it to work in your network. The first variable is called *tftp*. You must set this value to your TFTP server's IP address. The second variable you need to change is *workingdir*, which must contain the name of the directory that holds the list of routers, the file of configuration changes, and where the script will put its report.

The script is written in the Expect language and requires Expect to be loaded on your server and available in the */usr/local/bin* directory. For more information on the Expect language, see Appendix A or *Exploring Expect* (O'Reilly).

The script expects to find two files in the working directory. The first, *RTR_LIST*, contains a list of router names, with one name on each line. The second file, *NEWCONFIG*, contains all of the required configuration changes. We also recommend that you put the configuration command *end* on the last line of the *NEWCONFIG* file to avoid the error messages that we mentioned in Recipe 1.1.

Here is a typical example of the sort of things you might put in the configuration file:

```
Freebsd% cat NEWCONFIG
enable secret cisco
end
```

Using this file, the script will log into all of the routers and change the enable secret password. You could put any number of commands in this configuration file, but because the exact same set of configuration commands will be downloaded into every router, you should never change anything that is unique to a particular router (such as an IP address) this way. This method is perfect for changing passwords, SNMP community strings, access lists, and other things that can be the same on all routers.

The script will copy the *NEWCONFIG* file into the */tftpboot* directory so it can use TFTP to transfer it to each router. It is a good idea to ensure that your server's TFTP is working correctly before launching the script.

When the script finishes, it creates a status file called *RESULT* in the working directory. This status file contains detailed status reports of what happened on each router. The easiest way to see a list of the routers that the script failed to change is to use the Unix *grep* command to search for the keyword "fail":

```
Freebsd% grep fail RESULT
toronto - enable login failed
boston  - telnet failed
test    - enable login failed
frame   - enable login failed
```

1.16.4 See Also

Recipe 1.1; Appendix A; *Exploring Expect*, by Don Libes (O'Reilly)

Top

Recipe 1.17 Extracting Hardware Inventory Information

1.17.1 Problem

You need an up-to-date list of the hardware configurations and IOS levels of all of your routers.

1.17.2 Solution

The Bourne shell script in Example 1-3 uses SNMP to extract useful version information from a list of routers. By default, the script stores this data in CSV format so that you can easily import it into a spreadsheet for analysis. No arguments are required or expected.

Example 1-3. inventory.sh

```
#!/bin/sh
#
#    inventory.sh -- a script to extract valuable information
#                    from a list of routers. (Name, Type, IOS version)
#
#
# Set behaviour
public="ORARO"
workingdir="/home/cisco"
#
LOG=$workingdir/RESULT.csv
infile=$workingdir/RTR_LIST
snmp="/usr/local/bin/snmpget  -v1  -c  $public"
#
while read device
do
  $snmp $device sysName.0 > /dev/null
  if [ "$?" = "0" ] ; then
     rtr=`$snmp $device .1.3.6.1.4.1.9.2.1.3.0 | cut -f2 -d\" `
     type2=`$snmp $device .1.3.6.1.4.1.9.9.25.1.1.1.2.3 | cut -f2 -d$ `
     ios=`$snmp $device .1.3.6.1.4.1.9.9.25.1.1.1.2.5 | cut -f2 -d$ `
     prot=`$snmp $device .1.3.6.1.4.1.9.9.25.1.1.1.2.4 | cut -f2 -d$ `
    echo "$device, $rtr, $type2, $ios, $prot" >> $LOG
  fi
done < $infile
```

1.17.3 Discussion

The *inventory.sh* script extracts hardware and IOS version information directly from the routers using SNMP. This ensures that the data is up-to-date. You can even automate this script to run periodically, ensuring that your inventory information is always accurate. In a large network, this is much easier than keeping track of this information manually.

By default, the script captures the device name, router type, IOS version, and IOS feature set from each router. It stores this gathered information in a CSV format file called *RESULT.csv*.

This script requires NET-SNMP to gather the information via SNMP. You can use a different SNMP package if you prefer, but then you will need to modify the syntax appropriately. The script expects to find the executable *snmpget* in the */usr/local/bin* directory. Again, if you keep this file in a different location, you will need to define the correct location in the *snmp* variable. For more information on NET-SNMP, see Chapter 17 or Appendix A.

Before running this script in your network, you will need to modify two variables. The first is the *public* variable. This value must contain your read-only SNMP community string. The script assumes that you have the same community string on all of the routers in the list. The second variable that you will need to set is *workingdir*, which must contain the name of the directory that you wish to run the script from.

Finally, you will need to build a file called *RTR_LIST* that contains the names of all of your routers, with one name on each line. The script expects to find this file in the working directory.

The output of the script is a CSV file, which you can import into a spreadsheet to analyze and sort the results as required. Table 1-4 shows an example of the script's output as it might look in a spreadsheet.

Table 1-4. Output results from the inventory.sh script

Router	Router Name	Type	IOS version	IOS feature set
Toronto	toronto	C2600	12.2(13)	IP|FIREWALL 2 PLUS 3DES
Boston	boston	C3620	12.2(13)	IP|3DES PLUS
Newyork	newyork	C3620	12.2(13)	IP|3DES PLUS
Tampa	tampa	C2500	12.0(25)	IP PLUS
Sanfran	sanfran	C7200	12.0(12a)	IP|IPX|VLAN

1.17.4 See Also

Chapter 17; Appendix A

Top

Recipe 1.18 Backing Up Router Configurations

1.18.1 Problem

You need to download all of the active router configurations to see what has changed recently.

1.18.2 Solution

The Perl script in Example 1-4will automatically retrieve and store router configuration files on a nightly basis. By default, it will retain these configuration files for 30 days. The script should be run through the Unix *cron* utility to get the automatic nightly updates, but you can also run it manually if required. No arguments are required or expected.

Example 1-4. backup.pl

```
#!/usr/local/bin/perl
#
#       backup.pl -- a script to automatically backup a list of
#                    router configuration files on a nightly basis.
#
#
# Set behaviour
$workingdir="/home/cisco/bkup";
$snmprw="ORARW";
$ipaddress="172.25.1.1";
$days="30";
#
#
$rtrlist="$workingdir/RTR_LIST";
$storage="$workingdir/storage";
$latest="$storage/LATEST";
$prev="$storage/PREV";
if (! -d $storage) {mkdir ($storage, 0755)};
if (! -d $prev) {mkdir ($prev, 0755)};
if (! -d $latest) {mkdir ($latest, 0755)};
($sec, $min, $hr, $mday, $mon, $year, @etc) = localtime(time);
$mon++;  $year=$year+1900;
$today1=sprintf("%.4d_%.2d_%.2d", $year, $mon, $mday);
$today="$storage/$today1";
system("cp -p $latest/* $prev/");
unlink <$latest/*>;
mkdir ($today, 0755);

open (RTR, "$rtrlist") || die "Can't open $rtrlist file";
open (LOG, ">$workingdir/RESULT") || die "Can't open $workingdir/RESULT file";
```

```
print LOG "Router Configuration Backup Report for $year/$mon/$mday\n";
print LOG "= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =\n";
print LOG "Device Name                          Status\n";
print LOG "= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =\n";
while (<RTR>) {
  chomp($rtr="$_");
   $oid=".1.3.6.1.4.1.9.2.1.55.$ipaddress";
  $snmpset ="/usr/local/bin/snmpset -v1 -c $snmprw -t60 -r2 $rtr";
  $rtrfile="/tftpboot/$rtr.cfg";
  unlink $rtrfile;
  open (CFG, ">$rtrfile"); print CFG " ";close CFG;
  chmod 0666, $rtrfile;
  chop ($status=`$snmpset $oid s $rtr.cfg`);
  $status=~/.+ = "(.+)".*$/;
  if($1 eq "$rtr.cfg") {
     if( -z "$rtrfile" ) {
        $result="not ok (File empty)";
        unlink $rtrfile;
     }
     else {
        $result="ok";
        chmod 0444, $rtrfile;
        system("mv $rtrfile $latest");
     }
  }
  else {
     $result="not ok";
     unlink $rtrfile;
  }

printf LOG ("%-28s        %-28s\n", $rtr,$result);

}
system ("cp -p $latest/*cfg $today");
$time=$days*86400;
print "$time\n";
($sec, $min, $hr, $mday, $mon, $year, @etc) = localtime(time-$time);
$mon++;  $year=$year+1900;
$rmdir=sprintf("%s/%.4d_%.2d_%.2d",$configs,  $year,  $mon,  $mday);
system ("rm -r -f $storage/$rmdir");
```

1.18.3 Discussion

As we mentioned earlier in the chapter, it is extremely important to make regular backup copies of your router configuration files. However, as the size of your network grows, it becomes quite tedious to maintain a useful archive of these backups. This script automates the task of collecting and storing router configuration files on a Unix-based TFTP server.

This script will maintain 30 days worth of configuration files. We have found that this is a reasonable length of time, allowing engineers to recover router configuration files that are up to one month old.

However, if you prefer, you can change the *$days* variable to increase or decrease how long the script will store these files before deleting them. If you increase the length of time that the server must store these files, it will obviously increase the amount of disk space you need to hold the extra configuration files. But router configuration files are generally quite small, so this is usually not a serious problem unless you support thousands of routers.

Before executing this script, you will need to modify a few variables. First, the *$workingdir* variable should contain the name of the directory in which the server will run the script. Second, the *$snmprw* variable must contain your SNMP read-write community string. Please note that the read-only community string will not allow you to copy a configuration file; you must use the read-write string. The other variable you need to change is *$ipaddress* , which should contain the IP address of your TFTP server.

The script is written in Perl, and it makes a few system calls out to Bourne shell commands. The script expects to find the Perl executable in the */usr/local/bin* directory. The script is also dependent on NET-SNMP and it expects to find the executable *snmpset* in the */usr/local/bin* directory as well. If these files are in different locations on your local system, you will need to modify these paths. See Appendix A for more information on Perl and NET-SNMP.

Finally, you will need a file called *RTR_LIST* that contains the list of router names. This file must be in the working directory.

As we mentioned earlier, you should run this backup script from the Unix *cron* utility on a nightly basis. This ensures that you have an up-to-date backup of your configuration files. We recommend launching this script during the off-hours since it does generate traffic across your network, as well as causes a small amount of CPU loading on the routers. Here is an example *crontab* entry to start the script every night at 1:30 A.M.:

```
30 1 * * * /home/cisco/bkup/backup.pl
```

When the script runs, it creates a new directory called *storage* under the working directory. Under this directory, the script creates several subdirectories, including *LATEST* , *PREV* , and dated directory names (such as *2003_01_28*). The *LATEST* directory always contains the most up-to-date router configuration files, and you can find the previously stored version of each router's configuration in the *PREV* directory. The dated directories contain all of the router configuration files that were captured on the date indicated in the directory name.

You can use the Unix *diff* command to see what changes have occurred on a given router.

Finally, the script creates a nightly status report that it stores in a file called *RESULT* in the working directory:

```
Freebsd% cat RESULT
Router Configuration Backup Report for 2003/1/28
= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
```

```
Device Name                          Status
= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
toronto                              ok
boston                               not ok
test                                 ok
frame                                ok
```

With a slight modification, you can configure the script to email this report to the responsible engineer. However, since each different Unix flavor uses a different mail program, we chose not to include it here in the interest of compatibility. On a Solaris server, for example, you could add the following line to the bottom of the script to mail this report:

```
system ("/usr/ucb/mail -s \"Config Report for $today1\" `/bin/cat $mail` <
$workingdir/RESULT");
```

In this case, you would need to define the variable *$mail* to be an email distribution list for the report. For other Unix or Linux variants, please consult your manpages for more information on your local *mail* program.

1.18.4 See Also

Recipe 1.6 ; Appendix A

Top

Chapter 2. Router Management

Top

Introduction

Like the previous chapter, this chapter also looks at system management issues on the router. So far we've looked primarily at general system administration issues such as filesystem management, but here we will discuss management and tuning issues related to router performance. You'll also learn some of the techniques needed to deal with disaster scenarios, such as how to create exception dumps.

Cisco's IOS supports a variety of special purpose protocols and services. Some of these are useful for network management and administration, while others are more useful for testing purposes. One of the handiest features is the Cisco Discovery Protocol (CDP), which allows you to see useful information about the Layer 2 connections between Cisco devices. This chapter shows how to use CDP and covers some of its well-known security problems.

Disabling is often the best strategy for several other services. Some, like the HTTP management interface and various test protocols (lumped together under the title of the TCP and UDP "small servers"), serve no real purpose in most production networks and are disabled by default. Others, like DNS, do have useful functions and are enabled by default.

We will discuss several important administrative features such as different methods for handling the hostnames of other network devices and command aliases to make complex commands easier to remember and type. The chapter concludes with a set of four useful scripts for gathering important information from your network devices.

Top

Recipe 2.1 Creating Command Aliases

2.1.1 Problem

You want to create aliases for commonly-used or complex commands.

2.1.2 Solution

You can create command aliases on your router with the *alias* command:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#alias exec rt show ip route
Router1(config)#alias exec on show ip ospf neighbor
Router1(config)#end
Router1#
```

2.1.3 Discussion

Unix system administrators have been using command aliases for many years to help reduce typing and save time. These shortcut commands allow you to reduce long or complex command sequences down to a few simple characters. This is most useful for extremely common commands, or for those that are complex or difficult to remember. You can create an alias for any command, including some or all of its associated keywords or variables.

Here we have created the alias *rt* for one of the most common commands that we use every day, *show ip route*:

```
Router1(config)#alias exec rt show ip route
```

We can now use this simple two-letter command to display the routing table, saving time and typing:

```
Router1#rt
Codes: C - connected, S - static, I - IGRP, R - RIP, M - mobile, B - BGP
       D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
       E1 - OSPF external type 1, E2 - OSPF external type 2, E - EGP
       i - IS-IS, L1 - IS-IS level-1, L2 - IS-IS level-2, ia - IS-IS inter area
       * - candidate default, U - per-user static route, o - ODR
       P - periodic downloaded static route
```

```
Gateway of last resort is 172.25.1.1 to network 0.0.0.0

S     192.168.10.0/24 [1/0] via 172.22.1.4
      172.16.0.0/24 is subnetted, 1 subnets
C        172.16.2.0 is directly connected, FastEthernet0/0.2
      172.20.0.0/16 is variably subnetted, 3 subnets, 3 masks
O        172.20.10.0/24 [110/74] via 172.20.1.2, 00:52:55, Serial0/0.2
C        172.20.1.0/30 is directly connected, Serial0/0.2
O        172.20.100.1/32 [110/65] via 172.20.1.2, 00:52:55, Serial0/0.2
      172.22.0.0/16 is variably subnetted, 2 subnets, 2 masks
D       172.22.0.0/16 is a summary, 20:31:03, Null0
C        172.22.1.0/24 is directly connected, FastEthernet0/1
Router1#
```

The key to choosing a good alias command name is to pick something that is short and easy to remember. Of course, it is critical to select an alias that does not conflict with an existing command. In our example, we choose *rt* as a short and memorable mnemonic for "route table." This abbreviation does not conflict with any existing IOS command.

You can also use a command alias as part of a longer command. For example, we could use our *rt* alias to shorten the command *show ip route 172.16.2.0*.

```
Router1#rt 172.16.2.0
Routing entry for 172.16.2.0/24
  Known via "connected", distance 0, metric 0 (connected, via interface)
  Routing Descriptor Blocks:
  * directly connected, via FastEthernet0/0.2
      Route metric is 0, traffic share count is 1
Router1#
```

Command aliases are most effective if you use them consistently among all of the routers that you manage. Otherwise, you'll have to remember a different set of alias commands for each group of devices. If you want to use this feature, we recommend that the entire network management team work together to develop a standard set of aliases before implementing them. We also recommend keeping the aliases simple. And, above all, resist the urge to alias every possible command. Instead, create aliases for only the most common commands.

Command aliases are also useful for scripting. You can build a script to perform a task on a router that might be slightly different on each router. For example, suppose you want to clear the counters of a particular access list on a weekly basis. But, some of your routers use a different access list number. You can simply build an alias with the same name on each router, but make the actual commands represented by the alias appropriate to each individual router. Finally, you can build a script to issue the command alias and automate what would otherwise be an extremely onerous task.

The *show aliases* command displays all of the command aliases configured on the router:

```
Router1#show aliases
Exec mode aliases:
```

```
  h                    help
  lo                   logout
  p                    ping
  r                    resume
  s                    show
  u                    undebug
  un                   undebug
  w                    where
  rt                   show ip route
  on                   show ip ospf neighbor

Router1#
```

If you type this command on any router, you will see that Cisco implements several command aliases by default.

Top

Recipe 2.2 Managing the Router's ARP Cache

2.2.1 Problem

You want to adjust the ARP table timeout value.

2.2.2 Solution

To modify the ARP timeout value, use the *arp timeout* configuration command:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#interface Ethernet0
Router1(config-if)#arp timeout 600
Router1(config-if)#end
Router1#
```

2.2.3 Discussion

Every LAN device has an Address Resolution Protocol (ARP) cache. This is a table that the device uses to map Layer 2 MAC addresses to Layer 3 IP addresses. Without this mapping, the device could build its IP packets, but not the Layer 2 frames to carry them.

Devices discover the information in the ARP cache dynamically. If a device needs to send a packet to an IP destination, and it doesn't have a corresponding MAC address, it sends out a broadcast ARP request packet. This packet reaches every device on the LAN segment. The device that "owns" the IP address in question sends back an ARP response packet to complete the process.

Many LAN devices also automatically send a *gratuitous ARP* packet when they first connect to the network. A gratuitous ARP is a broadcast packet that is effectively an unsolicited ARP response. Every device on the LAN segment will receive this packet so that it can update its ARP cache in case there is ever a need to talk to this new device.

The ARP request and response process obviously takes time to complete, introducing a delay in packet processing. Furthermore, because the ARP request packets are broadcasts, they go to every device on the LAN segment and interrupt whatever that device was doing. If there are too many of these packets on the segment, it can cause traffic congestion and CPU loading on the connected devices.

To keep the ARP traffic down, all IP devices maintain a cache of these ARP entries. Old entries that

are no longer valid need to be periodically removed. The router needs to flush out old ARP cache entries faster in environments where devices frequently change their address, such as when there are very short DHCP lease times. In some cases there are so many devices that the ARP cache table becomes unwieldy, taking up too much memory or too much CPU time to support. Maintaining a balance between removing old invalid entries and keeping the amount of ARP traffic down is crucial.

By default, Cisco routers use an ARP cache timeout period of four hours. This means that if the router hasn't sent or received any packets with a particular address for the last four hours, it flushes the ARP entry from its cache. This period usually works well on Ethernet networks. However, there are special situations where you can improve network performance by adjusting this period.

The example in this recipe reduces the ARP timeout period to 600 seconds (10 minutes):

```
Router1(config-if)#arp timeout 600
```

Of course, you could just as easily use this command to increase the default ARP timeout period. In general we don't recommend using an ARP timeout period of less than about five minutes because a shorter period tends to cause too much CPU and network loading.

The *show ip arp* command prints out the current contents of the router's ARP cache:

```
Router1#show ip arp
Protocol  Address          Age (min)  Hardware Addr   Type   Interface
Internet  172.25.1.5             8   0001.9670.b780  ARPA   Ethernet0
Internet  172.25.1.7             -   0000.0c92.bc6a  ARPA   Ethernet0
Internet  172.25.1.1             9   0010.4b09.5700  ARPA   Ethernet0
Internet  172.25.1.3             2   0010.4b09.5715  ARPA   Ethernet0
Router1#
```

This output includes the IP address, age in minutes, MAC address, and the interface information for each ARP entry. The router resets the ARP age counter to zero whenever it sees valid traffic from the corresponding device. This ensures that the addresses of active devices are never flushed out of the cache, no matter how long they have been known.

You can specify a particular IP address with the *show ip arp* command. This can be useful when you are only interested in particular entries in a large cache table. On a large LAN core router, there could be hundreds or even thousands of ARP entries in the cache—far too many to scan by eye:

```
Router1#show ip arp 172.25.1.5
Protocol  Address          Age (min)  Hardware Addr   Type   Interface
Internet  172.25.1.5             2   0001.9670.b780  ARPA   Ethernet0
Router1#
```

The same command can also display the ARP information for a particular MAC address:

```
Router1#show ip arp 0010.4b09.5715
Protocol  Address          Age (min)  Hardware Addr   Type   Interface
Internet  172.25.1.3             3   0010.4b09.5715  ARPA   Ethernet0
```

```
Router1#
```

And you can even get a listing of ARP information for a particular router interface:

```
Router1#show ip arp Ethernet0
Protocol  Address          Age (min)  Hardware Addr   Type   Interface
Internet  172.25.1.5             4   0001.9670.b780  ARPA   Ethernet0
Internet  172.25.1.7             -   0000.0c92.bc6a  ARPA   Ethernet0
Internet  172.25.1.1             2   0010.4b09.5700  ARPA   Ethernet0
Internet  172.25.1.3             4   0010.4b09.5715  ARPA   Ethernet0
Router1#
```

When you are having an ARP problem or there are stale entries that you need to remove immediately, it can be useful to clear the entire cache. To manually clear the router's entire ARP cache, use the *clear arp* command:

```
Router1#clear arp
Router1#
```

Unfortunately, there is no way to remove a single ARP entry. If you need to manually clear an entry, you must erase the entire table. Doing this will cause a brief spike in ARP traffic as the router attempts to rebuild the ARP cache for the active devices, so use this command sparingly.

The *show interface* command includes information about the ARP timeout setting for a particular interface:

```
Router1#show interface Ethernet0
Ethernet0 is up, line protocol is up
  Hardware is Lance, address is 0000.0c92.bc6a (bia 0000.0c92.bc6a)
  Internet address is 172.25.1.7/24
  MTU 1500 bytes, BW 10000 Kbit, DLY 1000 usec, rely 255/255, load 1/255
  Encapsulation ARPA, loopback not set, keepalive set (10 sec)
  ARP type: ARPA, ARP Timeout 00:10:00
  <Removed for brevity>
```

Top

Recipe 2.3 Tuning Router Buffers

2.3.1 Problem

You want to change your default buffer allocations to improve router efficiency.

2.3.2 Solution

The router maintains two different sets of buffers; public buffers and interface buffers. The router uses these as temporary storage while processing packet data. You can tune the public buffer pools as follows:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#buffers big initial 100
Router1(config)#buffers big max-free 200
Router1(config)#buffers big min-free 50
Router1(config)#buffers big permanent 50
Router1(config)#end
Router1#
```

And you can adjust the interface buffer pools using a similar set of commands:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#buffers Ethernet0 initial 200
Router1(config)#buffers Ethernet0 max-free 300
Router1(config)#buffers Ethernet0 min-free 50
Router1(config)#buffers Ethernet0 permanent 50
Router1(config)#end
Router1#
```

2.3.3 Discussion

Before we start this discussion, we have three notes of caution on tuning buffers. First, adjusting your router's buffers is usually not necessary. Second, a poor set of buffer parameters can cause serious performance problems on your router and for traffic passing through the router. Third, if you do find that you need to adjust these parameters, the necessary adjustments will be unique to your network, and perhaps even to each router. Unfortunately, we can only offer general guidance.

The router maintains two different sets of buffers: public pools that the router can use for anything, and

interface specific pools that it can only use for processing packets on that interface.

The public buffers fall into several different pools, according to their size. These are shown in Table 2-1.

Table 2-1. Public buffer pools

Buffer size	Buffer pool name
104 bytes	Small buffers
600 bytes	Middle buffers
1536 bytes	Big buffers
4520 bytes	VeryBig buffers
5024 bytes	Large buffers
18024 bytes (default)	Huge buffers

Note that the huge buffers are 18,024 bytes by default. But, unlike the other public buffer pools, you can actually change the size of the buffers in this pool:

```
Router1(config)#buffers huge size 36048
```

You can configure any size between 18,024 and 100,000 bytes for your huge buffers. Since the router can only use memory in buffer-sized chunks, having extremely large buffers can be useful if you find that you need to manipulate extremely large packets. However, the default value of 18,024 should be large enough to handle the largest MTU values for all standard interface types. It's unlikely that you'll need to adjust this parameter. The remaining buffer sizes are all fixed and cannot be adjusted.

There are four different parameters that you can adjust on each of the public buffer pools:

```
Router1(config)#buffers big initial 100
Router1(config)#buffers big max-free 200
Router1(config)#buffers big min-free 50
Router1(config)#buffers big permanent 50
```

The first command sets the number of buffers of this type that the router will allocate at boot time. If this router is in an extremely high-traffic environment, it may take a while to allocate enough buffers to handle the load. You may find that the router has a few buffer failures right after booting. You can resolve this problem by increasing the number of initial buffers.

The second command uses the keyword *max-free* to set the maximum number of free buffers of this type (in this case, big) that the system should keep. In the router's normal functioning, it will see

periodic bursts of activity that may force it to allocate more buffers. When the burst is over, setting a relatively low value for *max-free* ensures that the router frees this extra memory to make it available for other purposes. But if you set it too low in an extremely busy environment, the router may not be able to allocate new buffers quickly enough to meet the demand.

In the third command, we have applied the *min-free* keyword to take care of the opposite side of the same problem. In order to help ensure that the router is able to handle the rising demand for packets, it will start allocating more system memory as soon as it finds that it has fewer than *min-free* more unused buffers of a particular type. If you specify a*min-free* value that is large enough, the router will be able to cope with any demands. But making the value too large will force the router to do additional work allocating additional buffers that it will never need.

In the final command we set the minimum number of buffers of this type using the keyword*permanent*. The router allocates this many buffers at boot time, and it will not return their memory to the general pool of memory. A good value for this parameter is high enough to reduce the amount of work that the router has to do allocating and trimming buffers, but not so high as to waste precious memory resources.

As you can see from the second example, the parameters for tuning the interface buffer pools are exactly the same as the ones we have just described for the public pools:

```
Router1(config)#buffers Ethernet0 initial 200
Router1(config)#buffers Ethernet0 max-free 300
Router1(config)#buffers Ethernet0 min-free 50
Router1(config)#buffers Ethernet0 permanent 50
```

The best way to tell whether your buffers need adjusting is to look at the output of the *show buffers* command:

```
Router1>show buffers
Buffer elements:
     498 in free list (500 max allowed)
     760166 hits, 0 misses, 0 created

Public buffer pools:
Small buffers, 104 bytes (total 50, permanent 50):
     50 in free list (20 min, 150 max allowed)
     265016 hits, 0 misses, 0 trims, 0 created
     0 failures (0 no memory)
Middle buffers, 600 bytes (total 25, permanent 25, peak 49 @ 1d09h):
     23 in free list (10 min, 150 max allowed)
     40749 hits, 10 misses, 30 trims, 30 created
     0 failures (0 no memory)
Big buffers, 1536 bytes (total 50, permanent 50):
     50 in free list (5 min, 150 max allowed)
     33780 hits, 0 misses, 0 trims, 0 created
     0 failures (0 no memory)
VeryBig buffers, 4520 bytes (total 10, permanent 10):
```

```
     10 in free list (0 min, 100 max allowed)
     0 hits, 0 misses, 0 trims, 0 created
     0 failures (0 no memory)
Large buffers, 5024 bytes (total 0, permanent 0):
     0 in free list (0 min, 10 max allowed)
     0 hits, 0 misses, 0 trims, 0 created
     0 failures (0 no memory)
Huge buffers, 18024 bytes (total 0, permanent 0):
     0 in free list (0 min, 4 max allowed)
     0 hits, 0 misses, 0 trims, 0 created
     0 failures (0 no memory)

Interface buffer pools:
Ethernet0 buffers, 1524 bytes (total 32, permanent 32):
     8 in free list (0 min, 32 max allowed)
     24 hits, 0 fallbacks
     8 max cache size, 8 in cache
     30963 hits in cache, 0 misses in cache
Serial0 buffers, 1524 bytes (total 32, permanent 32):
     4 in free list (0 min, 32 max allowed)
     54 hits, 3 fallbacks
     8 max cache size, 7 in cache
     172593 hits in cache, 32 misses in cache
Serial1 buffers, 1524 bytes (total 32, permanent 32):
     7 in free list (0 min, 32 max allowed)
     25 hits, 0 fallbacks
     8 max cache size, 8 in cache
     0 hits in cache, 0 misses in cache

Router1>
```

Let's zoom in on one of the public buffer pools to explain what the fields mean:

```
Small buffers, 104 bytes (total 50, permanent 50):
     50 in free list (20 min, 150 max allowed)
     265016 hits, 0 misses, 0 trims, 0 created
     0 failures (0 no memory)
```

This section looks at Small buffers, which are 104-byte chunks of memory. The router currently has allocated a total of 50 of these buffers; all 50 of them are permanent, meaning that the router will not attempt to return any of them to the pool of generally available memory.

In the second line, you can see that all 50 of these buffers are currently in the free list, meaning that they are all unused. The numbers 20 and 150 in this line are the min-free and max-free parameters that we discussed earlier.

In the third line, the number of hits indicates how many times the router has successfully allocated buffers from this pool. The number of misses indicates how many times the router successfully allocated a buffer from this pool, but in doing so, had to allocate additional buffers. The trims field counts the number of dynamically allocated buffers that the router has subsequently returned. The

created field shows how many buffers the router has actually created in response to miss events.

The last line shows serious problems, which are the only reasons that you should alter your buffer parameters. The failures field counts the number of times that the router has attempted to allocate a buffer and failed, causing it to drop the packet. The last field is labeled "no memory". This counts the number of times a failure occurred because the router had no memory from which to allocate additional buffers. This is an extremely serious problem, which is usually best treated by adding memory to the router.

It is also important to remember that if the router tries and fails to allocate a buffer from one pool, it will request a buffer from the next largest pool. So, for example, if the router is unable to get a Big buffer to handle a 1500-byte packet, it will use one from the VeryBig pool. This is why you can sometimes see buffer hits in the VeryBig pool, even if every interface on the router has an MTU of 1500 bytes. For this reason, it is a good idea to allocate a few permanent buffers from the pool larger than your highest MTU.

Now let's look at the interface buffers:

```
Ethernet0 buffers, 1524 bytes (total 32, permanent 32):
     8 in free list (0 min, 32 max allowed)
     24 hits, 0 fallbacks
     8 max cache size, 8 in cache
     30963 hits in cache, 0 misses in cache
```

This shows a similar set of values to what we just discussed for the public buffer pools, but there are a few differences. The first difference is the fallbacks field. This counts the number of times that the router has needed additional buffers on this interface, and has allocated them from the corresponding public buffer pool of the appropriate size. In this case, the Ethernet buffers are 1524 bytes, so the router would allocate additional buffers from the Big buffer public pool.

The router keeps a cache of buffers on each interface that are effectively in use whether there is data or not. This field varies somewhat depending on the hardware type. But, once again, you should watch out for misses. As long as the number of misses and fallbacks are low, there is no need to adjust the interface buffers.

We would like to offer one final caution on adjusting buffers. Always make sure to look at your router's free memory with the *show memory* command before and after making any adjustment:

```
Router1#show memory
                Head    Total(b)     Used(b)     Free(b)   Lowest(b)  Largest(b)
Processor     17DA4C    13112756     2308632    10804124    10577100    10663072
      I/O     E00000     2097152      336980     1760172     1740988     1759812
```

Keep close track of how much the free memory changes when you adjust your router's buffers. Both the processor and the I/O memory can be affected by these changes. If you inadvertently overallocate your buffers while trying to improve system performance, you may find that the router does not have enough

memory to operate properly when the load increases.

Top

Recipe 2.4 Using the Cisco Discovery Protocol

2.4.1 Problem

You want to see summary information about what is connected to your router's interfaces.

2.4.2 Solution

You can selectively enable or disable Cisco Discovery Protocol (CDP) on the entire router, or on individual interfaces:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#cdp run
Router1(config)#interface Serial0/0
Router1(config-if)#cdp enable
Router1(config-if)#exit
Router1(config)#interface FastEthernet0/0
Router1(config-if)#no cdp enable
Router1(config-if)#exit
Router1(config)#interface FastEthernet1/0
Router1(config-if)#cdp enable
Router1(config-if)#end
Router1#
```

2.4.3 Discussion

CDP is enabled by default on the router, and on all interfaces. If you have previously disabled it (as discussed in Recipe 2.5) and want to reenable CDP on the router, you can issue the *cdp run* global configuration command:

```
Router1(config)#cdp run
```

This turns on CDP processing on all supported interfaces by default. If you don't want to run CDP on a particular interface, you can use the *no cdp enable* command:

```
Router1(config)#interface Serial0/0
Router1(config-if)#no cdp enable
```

CDP is a Cisco proprietary protocol that allows Cisco devices to identify one another and exchange useful identifying information. The *show cdp neighbors* command gives a summary of information about adjacent devices that also happen to be running CDP:

```
Router1#show cdp neighbors
```

```
Capability Codes: R - Router, T - Trans Bridge, B - Source Route Bridge
                  S - Switch, H - Host, I - IGMP, r - Repeater

Device ID        Local Intrfce     Holdtme    Capability  Platform  Port ID
Router2          Ser 0/0            179          R        2621      Ser 0/1
Switch1          Fas 1/0            152        T S        WS-C2924  2/2
Router1#
```

As you can see, this output tells you the name and type of device of each neighbor, including the model number. It also includes both the interface on this router that connects to each neighbor, and the corresponding interface on the neighbor device.

The last of the devices listed is actually a Cisco Catalyst Ethernet switch. This points out one of the most useful features of CDP. While other mechanisms such as the ARP cache, routing protocols, or simple *ping* tests can tell you things about the Layer 3 neighbors, CDP gives you information about the Layer 2 neighbors. This is true even when a Layer 2 neighbor has no configured IP addresses.

You can see additional information about these neighboring devices by adding the *detail* keyword:

```
Router1#show cdp neighbors detail
-------------------------
Device ID: Router2
Entry address(es):
  IP address: 10.1.1.2
Platform: cisco 2621,  Capabilities: Router
Interface: Serial0/0,  Port ID (outgoing port): Serial0/1
Holdtime : 136 sec

Version :
Cisco Internetwork Operating System Software
IOS (tm) C2600 Software (C2600-IK9O3S-M), Version 12.2(13), RELEASE SOFTWARE (fc1)
Copyright (c) 1986-2002 by cisco Systems, Inc.
Compiled Tue 19-Nov-02 22:27 by pwade

advertisement version: 2

Device ID: Switch1
Entry address(es):
  IP address: 172.25.1.4
Platform: WS-C2924,  Capabilities: Trans-Bridge Switch
Interface: FastEthernet1/0,  Port ID (outgoing port): FastEthernet0/12
Holdtime : 116 sec

Version :
Cisco Internetwork Operating System Software
IOS (tm) C2900XL Software (C2900XL-C3H2S-M), Version 12.0(5)WC3b, RELEASE SOFTWARE
(fc1)
Copyright (c) 1986-2002 by cisco Systems, Inc.
Compiled Fri 15-Feb-02 10:14 by antonino
```

```
advertisement version: 2
Duplex: full

Router1#
```

This output tells you the IP addresses of the adjacent interfaces on the neighbor devices and gives details about the Cisco IOS or CatOS version.

Both of these neighbor devices support CDP Version 2, which Cisco introduced in IOS Version 12.0(3)T. It includes three new fields that are quite useful on LANs: VTP Domain Name, 802.1Q Native VLAN, and duplex configuration. As you can see in the previous output, the router and switch agree that they are operating at full duplex. Please refer to Chapter 16 for discussions of both 802.1Q and Ethernet Duplex configuration.

This new duplex option is extremely useful because the router and switch can now automatically detect duplex mismatches. We can demonstrate this ability by deliberately creating a duplex problem: in the following example, we change the switch's setting to half duplex for the port facing router. The router was able to detect the problem through CDP and issue the following log message:

```
Feb  6 11:36:11: %CDP-4-DUPLEX_MISMATCH: duplex mismatch discovered on
FastEthernet1/0 (not half duplex), with 003541987 (switch) FastEthernet0/12 (half
duplex).
```

CDP Version 2 is enabled by default on all IOS versions 12.0(3)T and higher. To globally disable Version 2 support on a router and allow only Version 1, issue the following global configuration command:

```
Router1(config)#no cdp advertise-v2
```

However, it is not entirely clear what purpose this would serve. We know of no interoperability problems between CDP Version 1 and Version 2. While there are security problems, which we will discuss in Recipe 2.5 , they are better addressed by disabling CDP altogether.

You can see global information about the router's CDP configuration with the *show cdp* command:

```
Router1#show cdp
Global CDP information:
        Sending CDP packets every 60 seconds
        Sending a holdtime value of 180 seconds
        Sending CDPv2 advertisements is  enabled
Router1#
```

Here you can see that this router sends out CDP advertisement packets every 60 seconds, which is the default. The holdtime parameter is the length of time that the router will wait to hear the next CDP advertisement from one of its neighbors. If it doesn't receive this advertisement packet within this time period, the router flushes the corresponding entry from its CDP neighbor table.

You can adjust these parameters globally for the entire router as follows:

```
Router1(config)#cdp timer 30
Router1(config)#cdp holdtime 240
```

Both of these commands accept an argument in seconds. The advertisement timer can have any value between 5 and 254 seconds, while the holdtimer must be between 10 and 255 seconds.

2.4.4 See Also

Recipe 2.5 ; Chapter 16

Top

Recipe 2.5 Disabling the Cisco Discovery Protocol

2.5.1 Problem

You don't want to allow adjacent devices to gain information about this router for security reasons.

2.5.2 Solution

You can disable CDP on a single interface using the *no cdp enable* interface configuration command:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#cdp run
Router1(config)#interface FastEthernet0/0
Router1(config-if)#no cdp enable
Router1(config-if)#end
Router1#
```

You can disable all CDP on the router with the *no cdp run* global configuration command:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#no cdp run
Router1(config)#end
Router1#
```

2.5.3 Discussion

CDP can be an extremely useful feature because it tells you so much about all of your neighboring devices. However, this can also represent a serious security problem. CDP packets are not encrypted, so someone who captures the CDP packets from a network segment as they pass between the routers can easily deduce a lot about your network architecture. If they can get access to the router either via Telnet or SNMP, they can use the CDP tables to discover the entire topology of your network at Layer 2 and 3, including all IOS levels, router and switch model types, and IP addressing. If somebody was armed with this information and a Cisco bug list, they could launch a very effective attack against your network.

For this reason, many network engineers choose to disable CDP throughout their networks. In general, if you need to disable CDP for security reasons, you should probably disable it globally on the whole router rather than on individual interfaces. If you disable CDP on a single interface, you will only

prevent people from intercepting the CDP advertisement packets. But the CDP table information is easily accessible through Telnet and SNMP, so valuable topology information is still vulnerable to probing.

We would like to clarify that the security risk is that somebody will launch a deliberate and focused attack against your network either from the inside or from a directly connected network. We strongly recommend disabling CDP on any routers that connect to external networks, particularly the public Internet. However, for purely internal networks, it is important to remember that you would be protecting yourself against people who are already physically connected to the network in some way. At this point you must balance the obvious usefulness of CDP against the risk of attack from people who probably have legitimate access to the network. Whether you disable CDP or not in this situation depends on how much you can trust your legitimate users not to launch a deliberate internal attack.

2.5.4 See Also

Recipe 2.4

Top

Recipe 2.6 Using the Small Servers

2.6.1 Problem

You want to enable or disable router services such as *finger*, *echo*, and *chargen*.

2.6.2 Solution

The *finger* application provides a remote way of seeing who is logged into the router. You can enable it with the *ip finger* global configuration command:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#ip finger
Router1#
```

Every Cisco router also has a set of small TCP and UDP server applications that are sometimes useful for testing purposes:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#service tcp-small-servers
Router1(config)#service udp-small-servers
Router1(config)#end
Router1#
```

2.6.3 Discussion

The *finger* command is a simple utility that allows you to do the equivalent of a *show users* command on a remote router. Unix computers generally have a standard *finger* program that you can run as follows:

```
Freebsd% finger @Router1
[Router1]

    Line       User       Host(s)              Idle       Location
    66 vty 0   kdooley    idle                 00:22:47   freebsd
    67 vty 1   ijbrown    idle                 1d07h      freebsd
*   68 vty 2              idle                 00:00:00   freebsd

    Interface  User        Mode                Idle       Peer Address
```

```
Freebsd%
```

You can also access the *finger* server by using the Telnet program to connect to TCP port 79. You can even do this from another router:

```
Router2#telnet 10.1.1.2 finger
Trying 10.1.1.2, 79 ... Open

    Line       User       Host(s)              Idle       Location
    66 vty 0   kdooley    idle                 00:24:14   freebsd
    67 vty 1   ijbrown    idle                 1d07h      freebsd
*   67 vty 1              idle                 00:00:00   10.2.2.2

  Interface    User       Mode                     Idle   Peer Address

[Connection to 10.1.1.2 closed by foreign host]
Router2#
```

Notice in both of these cases that the output includes not only the active users, but the finger process itself, which is indicated by an asterisk.

The *finger* protocol is defined in RFC 1288. It is disabled by default on Cisco routers. While *finger* is a convenient way to see who is logged in to a remote router without having to log in yourself to check, it can also represent a serious security problem. Not only does it display a set of valid login IDs that can be used to focus an attack, but it consumes a VTY line on the router, which can prevent legitimate access if done persistently. The *finger* protocol also has a checkered history—one of the first viral attacks to shut down large sections of the Internet (the infamous Morris Worm) exploited a bug in the original finger implementation.

For all of these reasons, we strongly recommend that you keep this protocol disabled on all of your routers. If it has been enabled for any reason, you can disable it as follows:

```
Router1(config)#no ip finger
```

Note that the *ip finger* command replaces the earlier *service finger* command found in many references:

```
Router1(config)#service finger
```

If you use this deprecated version of the command, the router automatically replaces it with the newer command.

Cisco routers also support a set of simple TCP and UDP applications that are relatively common standards for IP devices. In IOS levels 12.0 and higher, the TCP and UDP small servers are disabled by default. In earlier IOS levels, they are enabled by default.

In general, we find that the small servers are only marginally useful, and recommend disabling them when you are not actively using them for testing purposes. These servers listen for incoming packets from any source, which makes them vulnerable to network DoS attacks. Usually, the attacks simply

exploit the fact that the TCP servers will accept a connection from any device that requests one. If a hostile user sends a stream of TCP SYN packets to one of these ports, the router must respond to it and devote internal resources to keeping the session active. This can use up router resources.

The UDP servers are also potentially dangerous because a hostile user can spoof the source address in the packet to force your router to send a barrage of response packets to a third party. A similar attack could potentially be launched using the TCP servers, because the router will respond to any TCP SYN packet with a SYN ACK. Another network device could find itself unable to cope with receiving a barrage of unsolicited SYN ACK packets.

Therefore, we recommend disabling these services except when you specifically need to use one of them:

```
Router1(config)#no service tcp-small-servers
Router1(config)#no service udp-small-servers
```

However, with these cautions, the small servers do have legitimate uses.

The TCP and UDP small servers are shown in Table 2-2. The router implements both TCP- and UDP-based versions of each of these server functions, on the same port numbers. These are all well-known ports and commonly implemented applications. They are usually used for testing purposes.

Table 2-2. TCP and UDP small servers

Port number	Common name	RFC	Description
7	echo	RFC 862	The server process responds to any client input by sending back the identical data.
9	discard	RFC 863	The server process discards any data sent by the client.
13	daytime	RFC 867	The server responds with the current time and date, then closes the session.
19	chargen	RFC 864	The server sends a constant stream of ASCII characters to the client.

The easiest way to see what these functions do is to simply try them. The TCP versions are easier to demonstrate because you can use the standard Telnet application: just tell it to connect to a different TCP port number.

The echo function responds to whatever you type by sending back the same data:

```
Freebsd% telnet Router1 echo
Trying 172.25.25.1...
Connected to Router1.
Escape character is '^]'.
It gives a very echo to the seat where love is thron'd
It gives a very echo to the seat where love is thron'd
^]
telnet> quit
Connection closed.
Freebsd%
```

In its UDP version, the echo function merely copies the data segment of the packet and returns it to the sender.

The discard function is considerably less useful. The TCP version allows the client to establish a TCP session with the server, then ignores everything sent to it:

```
Freebsd% telnet Router1 discard
Trying 172.25.25.1...
Connected to Router1.
Escape character is '^]'.
Go off; I discard you: let me enjoy my private; go off.
^]
telnet> quit
Connection closed.
Freebsd%
```

The UDP version of this application listens for packets on UDP port number 9, but doesn't respond to them in any way.

The TCP version of the daytime server accepts a connection request, immediately sends a packet containing the current time and date in ASCII format, then disconnects the session:

```
Freebsd% telnet Router1 daytime
Trying 172.25.25.1...
Connected to Router1.
Escape character is '^]'.
Sunday, January 5, 2003 17:41:21-EST
Connection closed by foreign host.
Freebsd%
```

The UDP daytime server listens on UDP port number 13, and responds with a single packet containing the same ASCII time data as the TCP version. The daytime server is marginally useful for checking a clock, but other applications such as NTP are much more robust if you actually want to configure a reliable time service. We discuss NTP in Chapter 14.

The character generation (*chargen*) function is probably the most useful of the TCP small servers. As soon as you make a connection to this port number, the router will start sending a continuous stream of data back to the client. We have often used this feature as a sort of poor-man's traffic generator to

investigate network loading issues:

```
Freebsd% telnet Router1 chargen
Trying 172.25.25.1...
Connected to Router1.
Escape character is '^]'.
!"#$%&'(    )*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefg
!"#$%&'(    )*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefgh
"#$%&'(    )*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghi
#$%&'(    )*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghij
$%&'(    )*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijk
%&'(    )*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijkl
<similar lines deleted>
^]
telnet> quit
Connection closed.
Freebsd%
```

The UDP version of the *chargen* server listens for a UDP packet on the well-known port number 19, then generates a single response packet back to the sender. This response packet contains a random number between 0 and 512 bytes of arbitrary character data.

2.6.4 See Also

Chapter 14; RFC 1288, RFC 862, RFC 863, RFC 864, and RFC 867; Twelfth Night: Or, What You Will by William Shakespeare

---

Top

Recipe 2.7 Enabling HTTP Access to a Router

2.7.1 Problem

You want to configure and monitor your router using a browser interface.

2.7.2 Solution

Cisco includes an HTTP server in the IOS. You can enable this feature on a router and then use any standard web browser instead of Telnet to access the router:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#access-list 75 permit 172.25.1.1
Router1(config)#access-list 75 deny any
Router1(config)#ip http server
Router1(config)#ip http access-class 75
Router1(config)#end
Router1#
```

2.7.3 Discussion

After configuring this feature on a router, you can then connect to the router from a standard web browser. For example, using the Lynx text-based web browser, the router's home page looks like this:

```
                                                    Router1 Home Page

                                        Cisco Systems

Accessing Cisco 2621 "Router1"

        Telnet - to the router.

        Show interfaces - display the status of the interfaces.
        Show diagnostic log - display the diagnostic log.
        Monitor the router - HTML access to the command line interface at
                level 0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15

        Connectivity test - ping the nameserver.

        Show tech-support - display information commonly needed by tech
                support.
```

```
        QoS Device Manager - Configure and monitor QoS through the web
                interface.
    _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

  Help resources

    1. CCO at www.cisco.com - Cisco Connection Online, including the
       Technical Assistance Center (TAC).
    2. tac@cisco.com - e-mail the TAC.
     3. 1-800-553-2447 or +1-408-526-7209 - phone the TAC.
    4. cs-html@cisco.com - e-mail the HTML interface development group.
```

The bold words are links that allow you to execute IOS EXEC commands. For example, the Showinterfaces link will run the *show interfaces* command and display the result on your browser. You can even use the browser to configure the router. If you select one of the command-line interface level options, it will give you access to all of the EXEC commands at the corresponding authorization level. Please refer to Chapter 3 for more information about these user authorization levels.

This option for accessing a router has been available since IOS level 11.2. However, there was an extremely serious bug in the feature that was fixed in IOS level 12.1(5). This bug would cause the router to crash if the user issued a relatively simple typographical error. If a Telnet user types a question mark as part of a command, the router will respond with a list of valid options for this command. However, including a question mark in a URL would cause the router to crash. Since even a legitimate user could easily make this mistake, we strongly recommend against using the feature in any IOS levels before 12.1(5).

In more recent IOS versions, this web interface is no more or less secure than Telnet access to the router's EXEC command-line interface. You still need to supply the same valid user authentication information to connect using a browser. In Chapter 3 and Chapter 4 we will discuss different authentication methods, such as AAA, that you can use with Telnet. All these methods are also available for HTTP, and you can configure the one you want using the *authentication* keyword. For example, you can configure the HTTP server to use AAA authentication:

```
Router1(config)#ip http authentication aaa
```

You can even restrict which devices are permitted to access the router's web interface using the *access-class* keyword. In the example, we have told the router to restrict access to the router's web server based on access list number 75, which allows only one workstation IP address:

```
Router1(config)#access-list 75 permit 172.25.1.1
Router1(config)#access-list 75 deny any
Router1(config)#ip http access-class 75
```

We find that the Telnet command-line interface is much easier to use than the web interface. The only compelling use for this option that we have encountered is to allow first-level technical staff access to basic commands such as *show interfaces*.

2.7.4 See Also

Chapter 3

Top

Recipe 2.8 Using Static Hostname Tables

2.8.1 Problem

You want to create a static host lookup table on the router.

2.8.2 Solution

The *ip host* command lets you configure static host entries in the router:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#ip host freebsd 172.25.1.1
Router1(config)#ip host router2 10.1.1.1 172.22.1.4
Router1(config)#end
Router1#
```

2.8.3 Discussion

Many of the router commands will accept a hostname in place of an IP address. This helps to make the configuration files more readable, because it is much easier for humans to work with device names rather than IP addresses. However, the router still needs to translate these names into the IP addresses that it prefers to work with. Cisco supports two methods for resolving hostnames into IP addresses. You can use either static host entries as in this recipe, or DNS as in Recipe 2.9.

Static host entries are strictly local to the router. The router does not share this information with other routers or other devices. And, unlike DNS, static host entries are not dependent on any external services such as name servers. If you have both DNS and static host definitions, the router will use the static entries where possible. This allows you to override the normal DNS if you don't want to use it.

The biggest problem with static entries is, quite simply, that they are static and don't respond to IP address changes without manual intervention. The biggest advantage to static entries is that they don't depend on the reliability of any external servers. If you tie a critical function to a hostname instead of an IP address, that function may go away if the DNS server is temporarily unreachable.

For this reason, we strongly recommend using static host entries rather than DNS if you use hostnames in your router configuration instead of IP addresses.

In the example, the host called router2 has more than one IP address:

```
Router1(config)#ip host router2 10.1.1.1 172.22.1.4
```

When you associate multiple addresses with a single hostname like this, the router will attempt to connect to each address in the specified order. When building a static host entry for a neighboring router, you may wish to start with the loopback IP address followed by its other reachable IP addresses.

The *ip host* command also accepts a port number; this is the TCP port that the router will connect to when you use Telnet to reach the specified hostname. By default, the *telnet* command will initiate a connection to TCP port 23. In the following example, we will define a host named `mail`, and instruct the router to use port 25 (SMTP) when making connections to this device:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#ip host mail 25 172.25.1.1
Router1(config)#end
Router1#
```

Telneting now connects you to the SMTP process on the device:

```
Router1#telnet mail
Trying mail (172.25.1.1, 25 )... Open
220 freebsd.oreilly.com ESMTP Postfix
quit
221 Bye

[Connection to mail closed by foreign host]
Router1#
```

The router connects directly to the host's mail port, port 25. You can override the defined host port at the command prompt by including the required port number at the end of the *telnet* command:

```
Router1#telnet mail 25
```

You can use the *show hosts* command to get a complete list of all of the static host definitions:

```
Router1#show hosts
Default domain is not set
Name/address lookup uses static mappings

Host                          Port  Flags      Age Type   Address(es)
freebsd                       None  (perm, OK)  0   IP    172.25.1.1
router2                       None  (perm, OK)  0   IP    10.1.1.1
                                                          172.22.1.4
mail                          25    (perm, OK)  0   IP    172.25.1.1
Router1#
```

2.8.4 See Also

Recipe 2.9

Top

Recipe 2.9 Enabling Domain Name Services

2.9.1 Problem

You want to configure your router to use DNS to resolve hostnames.

2.9.2 Solution

To configure the router to use DNS to resolve hostnames, you need to specify a domain name and at least one name server:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#ip domain-lookup
Router1(config)#ip domain-name oreilly.com
Router1(config)#ip name-server 172.25.1.1
Router1(config)#ip name-server 10.1.20.5
Router1(config)#end
Router1#
```

2.9.3 Discussion

As we mentioned in Recipe 2.8, you can configure your router to use DNS to resolve hostnames. In fact, Cisco routers have DNS name resolution enabled by default. However, since there is no default name server, the router will attempt to use the local broadcast address (`255.255.255.255`) until you explicitly configure a proper name server. This means that the *ip domain-lookup* configuration command in the example is necessary only if someone has explicitly disabled DNS on the router.

After you configure the router with a valid name server, you can access any hostname known by your DNS server. For example, our DNS server exchanges information with the public Internet, so we can ping the Cisco web page by name:

```
Router1#ping www.cisco.com
Translating "www.cisco.com"...domain server (172.25.1.1) [OK]

Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 198.133.219.25, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 80/91/104 ms
Router1#
```

You can see that the router sent a DNS query to the name server (`172.25.1.1`) and asked it to translate the hostname www.cisco.com. The server responded with an IP address of `198.133.219.25`. The router then behaved as if we had simply asked it to ping this destination IP address instead of the hostname.

In this example, we have configured multiple name servers:

```
Router1(config)#ip name-server 172.25.1.1
Router1(config)#ip name-server 10.1.20.5
```

The router will send its queries to these servers in the order that we entered them. For example, suppose we tried to ping a fictitious host, *cookbook.oreilly.com*:

```
Router1#ping cookbook.oreilly.com
Translating "cookbook.oreilly.com"...domain server (172.25.1.1)(10.1.20.5)
% Unrecognized host or address, or protocol not running.

Router1#
```

As you can see, the router first sent this query to the name server at `172.25.1.1`. When this device was unable to resolve the name, the router resorted to the second name server, `10.1.20.5`. Ultimately the query failed because the hostname doesn't exist.

You can view the DNS configuration parameters with the *show hosts* command:

```
Router1#show hosts
Default domain is oreilly.com
Name/address lookup uses domain service
Name servers are 172.25.1.1, 10.1.20.5

Host                      Port  Flags      Age Type   Address(es)
www.cisco.com             None  (temp, OK) 0   IP     198.133.219.25
Router1#
```

This command displays the domain name, the name servers (in their order of preference), and recently resolved hostnames. The router keeps a name cache of recently resolved names to prevent unnecessary DNS lookups on successive attempts to the same host. The difference between these dynamically learned hosts and the statically configured ones that we saw in Recipe 2.8 is that the router will automatically flush the dynamic entries from the cache after a period of time. This time period is specified by the DNS server separately for each hostname; you cannot change it on the router.

The *ip domain-name* command allows you to specify your network's domain name:

```
Router1(config)#ip domain-name oreilly.com
```

When you configure a domain name like this, you can work with just the local hostname instead of the fully qualified domain name (FQDN). For example, you could type mail instead of mail.oreilly.com, and the router would resolve it correctly.

Some organizations use more than one domain name. You can configure the router to use multiple domain names by including several *ip domain-list* commands in the configuration. For example, we can configure the router to use a second registered domain name, *ora.com*:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#ip domain-list ora.com
Router1(config)#ip domain-list oreilly.com
Router1(config)#end
Router1#
```

If no domain list is present and you have specified a domain name, the router will use the given domain name. However, as soon as you configure a domain list, the router will ignore the domain name. This is why we included the original domain name, oreilly.com, in the domain list example.

Again, the order of the domain list entries is important because the router will use it to build the FQDN for its queries. For example, if you sent a query for the host named *mail*, the router would correctly find it in either domain. However, if a host named *mail* exists in both domains, the router will connect to *mail.ora.com* instead of *mail.oreilly.com*: this is the order specified by the domain list. You can still connect to *mail.oreilly.com* by entering the full domain name.

The *show hosts* command output includes the domain list:

```
Router1#show hosts
Default domain is oreilly.com
Domain list: ora.com, oreilly.com
Name/address lookup uses domain service
Name servers are 172.25.1.1, 172.25.1.3, 10.1.20.5

Host                      Port  Flags      Age Type   Address(es)
www.cisco.com             None  (temp, OK) 0   IP    198.133.219.25
freebsd                   None  (perm, OK) 0   IP    172.25.1.1
Router1#
```

2.9.4 See Also

Recipe 2.8

Top

Recipe 2.10 Disabling Domain Name Lookups

2.10.1 Problem

You want to prevent your router from trying to connect to your typing errors.

2.10.2 Solution

To prevent the router from attempting to resolve typing errors, use the *ip domain-lookup* command:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#no ip domain-lookup
Router1(config)#end
Router1#
```

You can also prevent the router from trying to resolve typing errors on routers that use DNS by changing the default EXEC behavior for unknown commands:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#line vty 0 4
Router1(config-line)#transport preferred none
Router1(config-line)#end
Router1#
```

2.10.3 Discussion

As we mentioned in Recipe 2.9, routers will attempt to resolve all hostnames using DNS by default. Unfortunately, if you don't configure a valid DNS name server, the router will send these queries to the local broadcast IP address, `255.255.255.255`. Querying a nonexistent name server is not only unproductive, but it can also be quite time consuming. If this happens in an interactive session, the router will not return the EXEC prompt until the query times out. This can be quite frustrating because, by default, the router interprets any unknown command as a hostname that you want to connect to and attempts to resolve any typing mistakes you enter on the command line:

```
Router1#pnig
Translating "pnig"...domain server (255.255.255.255)

Translating "pnig"...domain server (255.255.255.255)
 (255.255.255.255)
```

```
Translating "pnig"...domain server (255.255.255.255)
% Unknown command or computer name, or unable to find computer address
Router1#
```

As you can see, we accidentally mistyped the command *ping*. The router did not know this command, assumed that it must be the name of a foreign host, and attempted to resolve it. Everybody who has used a Cisco router for more than a few minutes is familiar with this problem: the annoyance of a typing error is compounded by having to wait several seconds for the name query to time out.

One easy way to prevent this from happening is to disable DNS lookups, as we did in our first example:

```
Router1(config)#no ip domain-lookup
```

This is an effective solution if you don't need to use DNS services on the router. With name resolution disabled, the router still interprets our typing mistakes as names of foreign hosts, but will only attempt to resolve names from the static host entries. These entries don't need to time out, so the router will return your prompt immediately and allow you to enter the command you intended to type:

```
Router1#pnig
Translating "pnig"
% Unknown command or computer name, or unable to find computer address
Router1#
```

Routers that are properly configured to use DNS services, as in Recipe 2.9, will also attempt to resolve your typing errors by default. Because there is a real server to respond to the request and definitively state that there is no such host, the delay is somewhat shorter. The router queries each of the configured name servers in order until it receives a response or gives up trying:

```
Router1#pnig
Translating "pnig"...domain server (172.25.1.1) (10.1.20.5)
% Unrecognized host or address, or protocol not running.

Router1#
```

This is still an extremely inefficient way of handling typing errors, and, if you need to use DNS, the solution given in our first example is not practical. Let's attack the problem from a different angle.

The router attempts to resolve typing errors because, by default, every VTY line has a preferred transport method of Telnet. This means that you can initiate a Telnet session by typing a hostname at the prompt. You don't need to explicitly issue the telnet command. Therefore, when we type in "pnig" the router interprets this as "telnet pnig". However, if we set the preferred transport method to "none," the router won't try to connect to a remote device unless we explicitly issue the *telnet* command:

```
Router1(config)#line vty 0 4
Router1(config-line)#transport preferred none
```

This solves the problem by preventing the router from misinterpreting our typos as hostnames in the first place:

```
Router1#pnig
         ^
% Invalid input detected at '^' marker.

Router1#
```

The router now interprets the typing error as an invalid command rather than a hostname. We recommend using this solution to the problem because it doesn't prevent you from using DNS.

2.10.4 See Also

Recipe 2.8; Recipe 2.9

Top

Recipe 2.11 Specifying a Router Reload Time

2.11.1 Problem

You want to set the router to automatically reload at a specified time.

2.11.2 Solution

You can set the router to reload after waiting a particular length of time with the *reload in* command:

```
Router1#reload in 20
Reload scheduled for 11:33:53 EST Sat Feb 1 2003 (in 20 minutes)
Proceed with reload? [confirm] <enter>
Router1#
```

The *reload at* command lets you specify a particular time and date when you want the router to reload:

```
Router1#reload at 14:00 Feb 2
Reload scheduled for 14:00:00 EST Sun Feb 2 2003 (in 26 hours and 44 minutes)
Proceed with reload? [confirm] <enter>
Router1#
```

If you set the router to reload at a specific time and date, then we highly recommend using an accurate time source to ensure that the router reloads when you think it will. For more information on time and time sources, see Chapter 14 .

2.11.3 Discussion

Usually, when you reload a router you want it to do so immediately. However, it can also be quite useful to specify a particular time to reload. For instance, reloading is the only way to fix badly fragmented memory on a router. But you almost certainly don't want to reload during production hours. This feature allows you to instruct the router to reload in the middle of the night or any other safe, low-traffic time.

Another excellent reason for using this delayed reload feature is to avoid locking yourself out of a router while making possibly dangerous configuration changes. There are many types of configurations changes—such as modifying access lists or routing configurations—that can isolate a router and prevent you from getting back in to fix the problem. But before you make the changes, you can instruct the router to reload itself in 15 minutes. If you lock yourself out of the router, you won't be able to save the running configuration to NVRAM. So when the router reloads, it comes up with the previous configuration. The bad configuration change is miraculously undone.

If it turns out that the new configuration is good, you can simply save it to NVRAM and cancel the reload. We will show how to cancel a scheduled reload in a moment.

The *reload in* command also allows you to specify a reason for the reload:

```
Router1#reload in 1:20 IOS Upgrade
Reload scheduled for 12:37:45 EST Sat Feb 1 2003 (in 1 hour and 20 minutes)
Reload reason: IOS Upgrade
Proceed with reload? [confirm] <enter>
Router1#
```

The command interprets any text that you enter after the reload time as the reason for reloading. Starting in IOS Version 12.2, the router records a log message whenever you issue the reload command. Included in this message are the time that the reload was requested, the reload time, the username of the person who requested it, and the reload reason:

```
Feb  1 11:17:47: %SYS-5-SCHEDULED_RELOAD: Reload requested for 12:37:45 EST Sat Feb 1
2003 at 11:17:45 EST Sat Feb 1 2003 by ijbrown on vty0 (172.25.1.1). Reload Reason:
IOS Upgrade.
```

You can also include a reason with the *reload at* command:

```
Router1#reload at 23:20 Feb 15 IOS Upgrade
Reload scheduled for 23:20:00 EST Sat Feb 15 2003 (in 124 hours and 48 minutes)
Reload reason: IOS Upgrade
Proceed with reload? [confirm] <enter>
Router1#
```

The *show reload* command displays information on any impending reloads:

```
Router1#show reload
Reload scheduled for 12:37:45 EST Sat Feb 1 2003 (in 1 hour and 19 minutes) by
ijbrown on vty0 (172.25.1.1)
Reload reason: IOS Upgrade
Router1#
```

You can cancel a scheduled reload with the *reload cancel* command:

```
Router1#reload cancel
Router1#

***
*** --- SHUTDOWN ABORTED ---
***
```

When you cancel a reload like this, the router sends a system broadcast message notifying any active users that the reload has been cancelled. Starting with IOS Version 12.2, the router will also create a logging message indicating that someone has cancelled a scheduled reload:

```
Feb  1 11:19:10: %SYS-5-SCHEDULED_RELOAD_CANCELLED: Scheduled reload cancelled at
```

```
11:19:10 EST Sat Feb 1 2003
Router1#
```

If you have scheduled a reload, the router send periodic shutdown notices to all active users. By default, the router sends these messages 1 hour, 30 minutes, 5 minutes, and 1 minute before reload. You can cancel the reload at any time, up until the router actually shuts itself down.

The shutdown messages look like this:

```
Router1#

***
*** --- SHUTDOWN in 1:00:00 ---
***

***
*** --- SHUTDOWN in 0:30:00 ---
***

***
*** --- SHUTDOWN in 0:05:00 ---
***

***
*** --- SHUTDOWN in 0:01:00 ---
***
Connection closed by foreign host.
```

2.11.4 See Also

Chapter 14

Top

Recipe 2.12 Creating Exception Dump Files

2.12.1 Problem

Your router is having serious problems and you need to create an exception dump to forward to Cisco's TAC.

2.12.2 Solution

To create an exception dump of a router's memory after a failure, you need to configure the *exception dump* command and tell the router how to automatically transfer this information to a server:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#ip ftp source-interface Loopback0
Router1(config)#ip ftp username ijbrown
Router1(config)#ip ftp password ijpassword
Router1(config)#exception protocol ftp
Router1(config)#exception region-size 65536
Router1(config)#exception dump 172.25.1.3
Router1(config)#end
Router1#
```

2.12.3 Discussion

This is the one recipe that we hope none of our readers ever need to use. The main reason for creating an exception dump of your router memory contents is to help Cisco's TAC to diagnose catastrophic software problems with one of your routers. When you have these types of extreme problems, the TAC will often request an exception dump of the router. We have included this recipe so that you'll know what to do if it ever becomes necessary.

An exception dump is a snapshot of the router's memory contents taken just before a software error forces the router to reload. The router must transfer this to a server because it is too much information to store in nonvolatile storage.

The dump actually creates two files: one of the main system memory, and one of the IO memory. The software engineers at Cisco can then use these two files to figure out what caused the software failure and (hopefully) create a fix for the next IOS release.

By default, the router uses TFTP to transfer the dump files. However, we strongly recommend using FTP

instead. If your router has to transfer more than 16MB of memory, most TFTP applications will fail, so FTP is the only option. FTP is also much more reliable than TFTP. Our example shows how to configure a router to use FTP to transfer the exception dump files from the router to the server. For more information on configuring the router to use FTP, see Recipe 1.14.

Exception dumps are prone to failure because the router doesn't attempt them until it has suffered a serious software failure. This software failure could corrupt your router's memory and make any further processing impossible—including creating the exception dump. You can greatly increase the chances of a successful exception dump by dedicating a small amount of memory to serve as a backup memory pool in case the main memory becomes corrupted:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#exception region-size 65536
Router1(config)#end
Router1#
```

You can define how much memory the router should dedicate to creating exception dumps. The default is 16,384 bytes, but we recommend increasing this to the maximum value of 65,536 bytes. This greatly increases the chances of a successful exception dump.

By default, the router creates two exception dump files named *hostname-core* and *hostname-coreiomem*. In our example the router name is Router1, so the two files created will be called *Router1-core* and *Router1-coreiomem*. You can change the default naming convention of the core files with the *exception core-file* command:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#exception core-file router5 compress
Router1(config)#end
Router1#
```

Note that we have included the optional keyword *compress* in this command. This option instructs the router to compress the core files before transferring them to the server. You can then uncompress the files on the server with the Unix *uncompress* command. We actually don't recommend using this feature though, because it adds extra CPU and memory load to a router that is already in trouble. Furthermore, we have found that this option doesn't seem to shrink the files enough to be useful. In some cases, it even causes the main core file to grow larger.

On the server side, you need to ensure that you have enough disk space to hold the two dump files. The size of these files will vary from router to router, depending on the amount of memory they contain. Expect the dump files to equal the total amount of memory installed in the router.

You can force the router to perform a core dump of its normal memory with the *write core* command. This command provides an excellent way to test whether everything is configured correctly before a catastrophic failure forces the real dump:

```
Router1#write core
Remote host [172.25.1.3]? <enter>
Base name of core files to write [Router1-core]? <enter>
Writing Router1-coreiomem
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
Writing Router1-core
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
Router1#
```

When we display these files on the server, you can see that their combined sizes are 48MB:

```
Freebsd% ls -la
drwxr-xr-x  3 ijbrown  ijbrown       512 Feb  1 13:50 ./
drwxr-xr-x  5 root     wheel         512 Feb  4  2002 ../
-rw-r--r--  1 ijbrown  ijbrown  46137344 Feb  1 13:54 Router1-core
-rw-r--r--  1 ijbrown  ijbrown   4194304 Feb  1 13:52 Router1-coreiomem
Freebsd%
```

To increase the chances of success, use a server that is as close as possible to the router. Time is of the essence during the creation of an exception dump, so forcing a router to write its core across a slow and congested WAN link decreases your chances of success.

2.12.4 See Also

Recipe 1.14

Top

Recipe 2.13 Generating a Report of Interface Information

2.13.1 Problem

You want to build a spreadsheet of active IP subnets for your network.

2.13.2 Solution

Keeping track of assigned IP subnets on a network is a vitally important but often tedious task. In large organizations, it can be extremely difficult to maintain accurate and up-to-date addressing information. The Perl script in Example 2-1 uses SNMP to automatically gather current IP subnet information directly from the routers themselves. The script creates an output file in CSV format so that you can easily import the information into a spreadsheet.

Example 2-1. netstat.pl

```
#!/usr/local/bin/perl
#
#      netstat.pl -- a script to build a detailed IP interface
#                    listing directly from a list of routers.
#
#Set behavior
$workingdir="/home/cisco/net";
$snmpro="ORARO";
#
$rtrlist="$workingdir/RTR_LIST";
$snmpwalk="/usr/local/bin/snmpwalk -v 1 -c $snmpro";
$snmpget="/usr/local/bin/snmpget -v 1 -c $snmpro";
open (RTR, "$rtrlist") || die "Can't open $rtrlist file";
open (CSV, ">$workingdir/RESULT.csv") || die "Can't open RESULT.csv file";
while (<RTR>) {
   chomp($rtr="$_");
    @ifIndex=`$snmpwalk $rtr .1.3.6.1.2.1.4.20.1.2`;
    @ipAddress=`$snmpwalk $rtr .1.3.6.1.2.1.4.20.1.1`;
    @ipMask=`$snmpwalk $rtr .1.3.6.1.2.1.4.20.1.3`;
    $arraynum=0;
   print CSV "\n$rtr\n";
   print CSV "Interface, IP-Address, Mask, MTU, Speed, Admin, Operational\n";
   for $ifnumber (@ifIndex) {
      chomp(($foo, $ifnum) = split(/= /, $ifnumber));
      $ifDescription=`$snmpget $rtr ifDescr.$ifnum`;
      $ifMTU=`$snmpget $rtr ifMtu.$ifnum`;
```

```
$ifSpeed=`$snmpget $rtr ifSpeed.$ifnum`;
$ifAdminstatus=`$snmpget $rtr ifAdminStatus.$ifnum`;
$ifOperstatus=`$snmpget $rtr ifOperStatus.$ifnum`;
chomp(($foo, $ipaddr) = split(/: /, $ipAddress[$arraynum]));
chomp(($foo, $mask) = split(/: /, $ipMask[$arraynum]));
chomp(($foo, $ifdes, $foo) = split(/"/, $ifDescription));
chomp(($foo, $mtu) = split (/= /, $ifMTU));
chomp(($foo, $speed) = split (/: /, $ifSpeed));
chomp(($foo, $admin) = split (/= /, $ifAdminstatus));
chomp(($foo, $oper) = split (/= /, $ifOperstatus));
if ( $speed > 3194967295 ) { $speed = 0 };
$admin =~ s/\(.*\)//;
$oper  =~ s/\(.*\)//;
if ( $oper eq "dormant" ) { $oper = "up(spoofing)"};
$speed = $speed/1000;
if ( $speed > 1000) {
   $speed = $speed/1000;
   $speed =~ s/$/ Mb\/s/;
      }
      else {
      $speed =~ s/$/ Kb\/s/;
      }
      print CSV "$ifdes,$ipaddr,$mask,$mtu,$speed,$admin,$oper\n";
      $arraynum++;
   }
}
close(RTR);
close(CSV);
```

2.13.3 Discussion

The *netstat.pl* script uses SNMP to gather IP subnet information from a list of routers. This ensures that the information is accurate and current. The script gathers all of the pertinent information about each router's IP interfaces and outputs this information as a CSV file.

The *netstat.pl* script requires Perl and NET-SNMP to be in the */usr/local/bin* directory. For more information on Perl or NET-SNMP, see Appendix A. If you keep these programs in a different location, you will need to modify the script appropriately.

Before using the script you must define two variables, *$workingdir* and *$snmpro*. The *$workingdir* variable must contain the name of directory where you will store the script and its input and output files. The variable *$snmpro* must contain the SNMP read-only community string for your routers. The script assumes that the same community string is valid on all devices.

The script systematically queries each router in a list, one after another. It expects to find this list in a file called *RTR_LIST*, located in the working directory. This router list should contain a single router name or IP address on each line with no comments or other information included. The results of the script are stored in a file called *RESULT.csv*, also located in the working directory.

You can then import the *RESULT.csv* file into a spreadsheet. The results will look similar to the example report shown in Table 2-3.

Table 2-3. An example output of the netstat.pl script

Detroit						
Interface	IP Address	Mask	MTU	Speed	Admin	Oper
Serial0/0	10.1.1.1	255.255.255.252	1500	768Kb/s	up	up
Loopback0	10.2.2.2	255.255.255.252	1514	0Kb/s	up	up
FastEthernet1/0	172.22.1.4	255.255.255.0	1500	100Mb/s	up	up
Ethernet0/0	172.25.1.8	255.255.255.0	1500	10Mb/s	down	down
Toronto						
Interface	IP Address	Mask	MTU	Speed	Admin	Oper
BRI0	10.1.99.55	255.255.255.0	1500	64Kb/s	down	down
Ethernet0	172.25.1.7	255.255.255.0	1500	10Mb/s	up	up
Loopback0	172.25.25.6	255.255.255.255	1514	0Kb/s	up	up
Boston						
Interface	IP Address	Mask	MTU	Speed	Admin	Oper
Serial0.1	172.20.1.2	255.255.255.252	0	28Kb/s	up	up
Ethernet0	172.20.10.1	255.255.255.0	1500	10Mb/s	up	up
Loopback0	172.20.100.1	255.255.255.255	1514	0Kb/s	up	up

The script captures information about every interface that has been configured with an IP address. This includes interfaces that are administratively or operationally down, loopback interfaces, HSRP addresses, and IP unnumbered interfaces. The script will not display any non-IP interfaces or subinterfaces.

Because this script uses only open standard SNMP MIBs, you can use it to extract IP interface information from almost any SNMP-enabled device, including non-Cisco equipment.

2.13.4 See Also

Appendix A

Top

Recipe 2.14 Generating a Report of Routing Table Information

2.14.1 Problem

You need to extract the IP routing table from one of your routers.

2.14.2 Solution

The script in Example 2-2, *rt.pl*, uses SNMP to extract the routing table from a specified router, and displays this information to STDOUT. The script expects to find a hostname or IP address of a router on the command line.

Example 2-2. rt.pl

```
#!/usr/bin/perl
#
#           rt.pl -- a script to extract the routing table
#                    from a router.
#
#Set behavior
$snmpro="ORARO";
#
$x=0;
$snmpwalk="/usr/local/bin/snmpwalk -v 1 -c $snmpro";
$snmpget="/usr/local/bin/snmpget -v 1 -c $snmpro";
chomp ($rtr=$ARGV[0]);
if ( $rtr eq "" ) {die "$0: Must specify a router\n"};
print "Destination\tMask\t\tNexthop";
print "\t\t Proto\tInterface\n";
@iftable=`$snmpwalk $rtr ifDescr`;
for $ifnum (@iftable) {
    chomp (($intno, $intname) = split (/ = /, $ifnum));
    $intno=~s/.*ifDescr\.//;
    $intname=~s/"//gi;
    $int{$intno}=$intname;
}
@ipRouteDest=`$snmpwalk $rtr ipRouteDest`;
@ipRouteMask=`$snmpwalk $rtr ipRouteMask`;
@ipRouteNextHop=`$snmpwalk $rtr ipRouteNextHop`;
@ipRouteProto=`$snmpwalk $rtr ipRouteProto`;
@ipRouteIfIndex=`$snmpwalk $rtr ipRouteIfIndex`;
#@ipRouteMetric1=`$snmpwalk  $rtr  ipRouteMetric1`;
```

```
for $intnum (@ipRouteIfIndex) {
    chomp (($foo, $int) = split (/= /, $intnum));
    chomp (($foo, $dest) = split (/: /, @ipRouteDest[$x]));
    chomp (($foo, $mask) = split (/: /, @ipRouteMask[$x]));
    chomp (($foo, $nhop) = split (/: /, @ipRouteNextHop[$x]));
    chomp (($foo, $prot) = split (/= /, @ipRouteProto[$x]));
     #chomp (($foo, $metr) = split (/= /, @ipRouteMetric1[$x]));
    $int1 = $int{$int};
     if ($int1 eq '') {$int1="Local"};
    $prot=~s/\(.*//; $prot=~s/ciscoIgrp/\(e\)igrp/;
     printf ("%-15s %-15s %-15s %7s %-25s\n",$dest, $mask, $nhop, $prot, $int1);
    $x++
}
```

2.14.3 Discussion

The *rt.pl* script is written in Perl and uses NET-SNMP to extract the routing table information via SNMP. Perl and NET-SNMP must be in the */usr/local/bin* directory. For more information on Perl or NET-SNMP, see Appendix A.

You must define the variable *$snmpro* to contain the SNMP read-only community string for the router before using this script:

```
Freebsd% ./rt.pl toronto
Destination     Mask            Nexthop         Proto  Interface
10.1.1.0        255.255.255.252 172.25.1.5       ospf   Ethernet0
10.2.2.2        255.255.255.255 172.25.1.5       ospf   Ethernet0
172.16.2.0      255.255.255.0   172.25.1.5       ospf   Ethernet0
172.20.0.0      255.255.0.0     172.25.1.5       local  Local
172.20.1.0      255.255.255.252 172.25.1.5       ospf   Ethernet0
172.20.10.0     255.255.255.0   172.25.1.5       ospf   Ethernet0
172.20.100.1    255.255.255.255 172.25.1.5       ospf   Ethernet0
172.22.0.0      255.255.0.0     172.25.1.5      (e)igrp Ethernet0
172.22.1.0      255.255.255.0   172.25.1.5       ospf   Ethernet0
172.25.1.0      255.255.255.0   172.25.1.7       local  Ethernet0
172.25.2.0      255.255.255.252 172.25.1.5      (e)igrp Ethernet0
172.25.25.1     255.255.255.255 172.25.1.5      (e)igrp Ethernet0
172.25.25.6     255.255.255.255 172.25.25.6      local  Loopback0
172.25.26.4     255.255.255.252 172.25.1.5      (e)igrp Ethernet0
172.25.26.5     255.255.255.255 172.25.1.5       ospf   Ethernet0
Freebsd%
```

The output from the script is relatively straightforward, except for static routes and directly connected routes, which require a little explanation.

For static routes, the output shows a value of "local" in the protocol field and an interface name of "Local." Directly connected routes also appear as "local" protocol information, but the interface name is the real interface associated with this route.

For example, the route `172.20.0.0  255.255.0.0` is a static route:

```
172.20.0.0      255.255.0.0     172.25.1.5        local Local
```

`172.25.1.0  255.255.255.0` is a directly connected route:

```
172.25.1.0      255.255.255.0   172.25.1.7        local Ethernet0
```

Since this script queries only open standard SNMP MIB value, you can use it to extract IP route information from most any SNMP-enabled device, including non-Cisco equipment.

2.14.4 See Also

Appendix A

Top

Recipe 2.15 Generating a Report of ARP Table Information

2.15.1 Problem

You need to extract the ARP table from one of your routers to determine the MAC address associated with a particular IP address or the IP address for a particular MAC address.

2.15.2 Solution

The script in Example 2-3, *arpt.pl*, extracts the ARP table for a specified router or IP address and displays the results to STDOUT. The script expects to find a hostname or IP address of a router on the command line.

Example 2-3. arpt.pl

```
#!/usr/local/bin/perl
#
#         arpt.pl -- a script to extract the ARP cache from a router.
#
#Set behavior
$snmpro="ORARO";
#
$snmpwalk="/usr/local/bin/snmpwalk -v 1 -c $snmpro";
$snmpget="/usr/local/bin/snmpget -v 1 -c $snmpro";
chomp ($rtr=$ARGV[0]);
if ( $rtr eq "" ) {die "$0: Must specify a router \n"};
@iftable=`$snmpwalk $rtr ifDescr`;
for $ifnum (@iftable) {
    chomp (($intno, $intname) = split (/ = /, $ifnum));
    $intno=~s/.*ifDescr\.//;
    $intname=~s/"//gi;
    $arpint{$intno}=$intname;
}
printf ("%-22.22s %-10.10s  %-25.25s\n", Address, MAC, Interface);
@atTable=`$snmpwalk $rtr .1.3.6.1.2.1.3.1.1.1`;
for $atnum (@atTable) {
    chomp (($atip, $atint) = split (/ = /, $atnum));
     $atip =~ s/.*atIfIndex\.[0-9]+\.1\.//;
     $atphys=`$snmpget $rtr atPhysAddress.$atint.1.$atip`;
    chomp(($foo, $phys) = split(/: /, $atphys));
    $phys=~s/ /-/gi; chop ($phys);
    $phys=~tr/A-Z/a-z/;
```

```
    $int=$arpint{$atint};
     printf ("%-15.15s %17.17s  %-25.25s\n", $atip, $phys, $int);
}
```

2.15.3 Discussion

The *arpt.pl* script extracts the ARP table from a specific router using SNMP and displays it to STDOUT. The script requires Perl and NET-SNMP and it expects to find both in the */usr/local/bin* directory. For more information on Perl or NET-SNMP, see Appendix A.

Be sure to set the SNMP read-only community string (contained in the variable *$snmpro*) before using this script:

```
Freebsd% ./arpt.pl toronto
Address                  MAC          Interface
172.22.1.1      00-01-96-70-b7-81  FastEthernet0/1
172.22.1.2      00-01-96-70-b7-81  FastEthernet0/1
172.22.1.3      00-01-96-70-b7-81  FastEthernet0/1
172.25.1.1       00-10-4b-09-57-00  FastEthernet0/0.1
172.25.1.5       00-01-96-70-b7-80  FastEthernet0/0.1
172.25.1.7       00-00-0c-92-bc-6a  FastEthernet0/0.1
172.25.1.254     00-00-0c-07-ac-01  FastEthernet0/0.1
172.16.2.1       00-01-96-70-b7-80  FastEthernet0/0.2
172.16.2.22      00-00-0c-07-ac-00  FastEthernet0/0.2
Freebsd%
```

The script creates a simple report including the IP address, MAC address, and interface name of each ARP entry. You can then use a search utility to locate specific devices by their IP or MAC addresses. For example, on a Unix server, you could pipe the output to the *grep* command, as follows:

```
Freebsd% ./arpt.pl toronto | grep 172.25.1.5
172.25.1.5       00-01-96-70-b7-80  FastEthernet0/0.1
Freebsd%
```

The ARP tables on core routers can be quite large, which makes locating a single ARP entry difficult. This script allows you to track down a particular device remotely. You could also use the *grep* utility to find the IP address of a particular known MAC address:

```
Freebsd% ./arpt.pl toronto | grep 00-10-4b-09-57-15
172.25.1.3       00-10-4b-09-57-15  FastEthernet0/0.1
Freebsd%
```

This script only queries open standard SNMP MIBS, so you can use it to extract ARP table information from almost any SNMP enabled device, even non-Cisco equipment.

2.15.4 See Also

Appendix A

Top

Recipe 2.16 Generating a Server Host Table File

2.16.1 Problem

You want to build a detailed host file containing the IP addresses and interface names of all of your routers.

2.16.2 Solution

The Perl script in Example 2-4, *host.pl*, builds a detailed host table that includes all of the IP addresses on each router in a list of devices. The script is written in Perl and requires NET-SNMP to extract data from the router list. No arguments are expected or required.

Example 2-4. host.pl

```
#!/usr/local/bin/perl
#
#         host.pl -- a script to build a detailed host file from
#                    information gathered from a router list.
#
#Set behavior
$workingdir="/home/cisco/net";
$snmpro="ORARO";
#
$rtrlist="$workingdir/RTR_LIST";
$snmpwalk="/usr/local/bin/snmpwalk -v 1 -c $snmpro";
$snmpget="/usr/local/bin/snmpget -v 1 -c $snmpro";
open (RTR, "$rtrlist") || die "Can't open $rtrlist file";
open (RESULT, ">$workingdir/RESULT") || die "Can't open RESULT file";
while (<RTR>) {
   chomp($rtr="$_");
   @ifIndex=`$snmpwalk $rtr ipAdEntIfIndex`;
   @ipAddress=`$snmpwalk $rtr ipAdEntAddr`;
    $rtr1=`$snmpget $rtr .1.3.6.1.4.1.9.2.1.3.0`;
    chomp(($foo, $RTR) = split (/"/, $rtr1));
    $arraynum=0;
    for $ifnumber (@ifIndex) {
       chomp(($foo, $ifnum) = split(/= /, $ifnumber));
       $ifDescription=`$snmpget $rtr ifName.$ifnum`;
       chomp(($foo, $ipaddr) = split(/: /, $ipAddress[$arraynum]));
       chomp(($foo, $ifdes) = split(/= /, $ifDescription));
       $name="$RTR-$ifdes";
```

```
        #$name=~s/\//-/;
        if ( $ifdes eq "Lo0" ) { $name=$RTR };
        print RESULT "$ipaddr\t\t$name\n";
        $arraynum++;
    }
}
close(RTR);
close(RESULT);
```

2.16.3 Discussion

Most organizations manually build a host table for their management server(s), with a single IP entry per router, usually the loopback IP address. This script automatically builds a host file that contains all known IP addresses for each router.

Here is an example of the output from the *host.pl* script:

```
10.1.1.1                miami-Se0/0
10.2.2.2                miami
172.20.6.8              miami-Se0/2
172.22.1.4              miami-Fa1/0
172.25.1.8              miami-Et0/0
10.1.1.2                toronto-Se0/1
172.20.1.1              toronto-Se0/0.2
172.22.1.1              toronto-Fa0/1
172.25.1.5              toronto-Fa0/0.1
172.25.2.1              toronto-Se0/0.1
172.25.25.1             toronto
172.25.26.5             toronto-Lo1
10.1.99.55              detroit-BR0
172.25.3.7              detroit-Et0
172.25.25.6             detroit
172.20.1.2              boston-Se0.1
172.20.10.1             boston-Et0
172.20.100.1            boston
```

This output is in the format required for a Unix */etc/hosts* file. The script extracts the IP address information via SNMP, then associates each address with the related router name and interface. The script also creates a primary host entry for each router using the address of the `loopback0` interface, but with no interface information in the hostname. In this example, you can reach the router located in Boston with hostname *boston* rather than the less intuitive *boston-Lo0*.

Having a detailed host file is useful for many reasons. Sometimes the router will send a message to your server, either using SNMP or *syslog*, and use the IP address of the interface instead of a main loopback interface. Also, having a detailed host file makes the output of a *traceroute* command much easier to understand:

```
Freebsd% traceroute miami
traceroute to miami (10.2.2.2), 64 hops max, 52 byte packets
```

```
 1  detroit-Et0       (172.25.3.7)  2.263 ms  2.210 ms  2.178 ms
 2   toronto-Fa0/0.1  (172.25.1.5)   3.042 ms  3.060 ms  3.846 ms
 3  boston-Se0.1      (172.20.1.2)  8.234 ms  8.245 ms  8.145 ms
 4  miami-Se0/2       (172.20.6.8)  9.893 ms  9.893 ms  9.432 ms
Freebsd%
```

This makes it much easier to decipher the path between the management station and the Miami router. Not only can we tell which routers lie along the path, but we can also clearly see which interfaces a packet sent along this path will use.

The script does not update the */etc/hosts* file directly. You may need to manually import the script's output file into your system's */etc/hosts* file. However, you can make this task easier by building a master file containing your normal host information and concatenating this master file together with the script output. This way you could even use the *cron* utility to automatically run this script and create a new up-to-date host file on a nightly basis.

Before the script will work, you must modify two variables. The *$workingdir* variable must be set to the directory that you will launch the script from. The *$snmpro* variable must be set to your SNMP read-only community string. The script assumes that you use the same read-only community string on all of your routers.

The script reads through a router list, and queries each device in sequence. It expects to find this list in a file called *RTR_LIST* in the working directory. The list can contain router names or IP addresses, with one entry per router, and one router per line. The script will extract the hostname directly from the router so you should also ensure that all of your routers are configured with unique hostnames. The results of the script are stored in a file called *RESULT*, contained in the working directory.

As a final note, we should mention that this script can generate hostnames that do not confirm to RFC 952, "DoD Internet Host Table Specification," which defines the official rules for hostnames. This is because the script can create hostnames with a forward slash (/) character in them, such as *miami-Se0/2*. This may cause problems for some applications, particularly if they use URL format addressing. For example, a query to http://miami-Se0/2 will clearly cause problems because the last character, "2", will be interpreted as a filename. However, most common applications such as *ping* and Telnet will accept this hostname without any complaints.

If you are a purist, or have applications that complain about these hostnames, we've included a line in the script that you can use to convert all of the slashes (/) to dashes (-). This line is currently commented out, but you can invoke it by simply removing the comment character (#) to change this line:

```
#$name=~s/\//-/;
```

to this:

```
$name=~s/\//-/;
```

2.16.4 See Also

RFC 952

Top

Chapter 3. User Access and Privilege Levels

Top

Introduction

Many network administrators do only the minimum when it comes to setting up user access to their routers. This is sufficient in networks where there are no serious security issues, and only a small number of people ever want or need to access the router. But, unfortunately, not every administrator can be quite so cavalier.

Most of the recipes in this chapter discuss methods for securing access to routers through important measures such as assigning usernames and passwords, controlling access-line parameters, handling remote access protocols, and affecting privileges of users and commands.

There are several important prerequisites for this discussion. You should understand what VTYs and access lines are. You should also have knowledge of user and command privilege levels. These topics are discussed in Chapters 4 and 13 ofCisco IOS In A Nutshell (O'Reilly).

We discuss best practices and provide a number of valuable recommendations in this chapter. We recommend referring to the National Security Agency (NSA) router security documents for more information. This extremely useful set of recommendations covers many different types of systems, including Cisco routers. You can download the Cisco section of this document from http://www.nsa.gov/snac/cisco.

Many examples in this chapter make limited use of Cisco's advanced authentication methodology called Authentication, Authorization, and Accounting (AAA). In this chapter, we will focus on purely local AAA implementations. We discuss AAA in more detail in Chapter 4, where we describe how to centralize these servers with TACACS+.

This chapter also contains three scripts written by the authors of this book. Two of these scripts are written in Perl, and the other is in Expect. For more information on these languages, refer to *Programming Perl* and *Exploring Expect* (both from O'Reilly). Appendix A includes information on obtaining copies of these packages and finding documentation for them.

Top

Recipe 3.1 Setting Up User IDs

3.1.1 Problem

You want to assign individual (or group) user IDs and passwords to network staff.

3.1.2 Solution

Use the following set of configuration commands to enable locally administered user IDs:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#username ijbrown password oreilly
Router1(config)#username kdooley password cookbook
Router1(config)#aaa new-model
Router1(config)#aaa authentication login default local
Router1(config)#end
Router1#
```

The *username* command also allows you to create usernames without passwords by specifying the *nopassword* keyword:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#username weak nopassword
Router1(config)#aaa new-model
Router1(config)#aaa authentication login default local
Router1(config)#end
Router1#
```

However, we strongly recommend against doing this because it can severely weaken the router's security.

3.1.3 Discussion

Enabling locally administered usernames overrides the default VTY password-based authentication system. When you enable the *aaa new-model* command, as shown in this recipe, the router immediately begins to prompt for usernames and passwords. Assigning unique usernames to individuals or groups provides accountability, as we will show later. The following example shows the login prompt for a router using local authentication:

```
Freebsd%telnet Router1
Trying 172.25.1.5...
Connected to Router1.
```

```
Escape character is '^]'.

User Access Verification

Username: ijbrown
Password: <password>

Router1>
```

The router prompts for the username as well as the password. Compare this to how the router behaves by default:

```
Freebsd%telnet Router2
Trying 172.25.1.6...
Connected to Router2.
Escape character is '^]'.

User Access Verification

Password: <password>

Router2>
```

Locally administered usernames work well in a small environment with a limited number of administrators. However, this method does not scale well to a large network with many administrators. Keeping usernames synchronized across an entire network can become quite daunting. Fortunately, Cisco also supports a centralized authentication system, which we discuss in Chapter 4 .

When you configure locally administered usernames, the router prompts for usernames on all lines, including the console and AUX ports, as well as the VTY ports used for Telnet sessions. To avoid locking yourself out of the router, you should always configure a *username* command before entering the AAA commands. It is also a good idea to use another session terminal to test the new authentication system before logging out of your original session. If you do accidentally lock yourself out of the router, you will need to follow the normal password-recovery procedures for your router type.

Enabling username support causes the router to associate certain functions with usernames. This provides accountability for each username by showing exactly who is doing what. For instance, the output of the *show users* command includes active usernames:

```
Router1>show users
  Line       User       Host(s)              Idle       Location
  66 vty 0     ijbrown    idle                 00:36:21   freebsd.oreilly.com
  67 vty 1     kdooley    idle                 00:00:24   server1.oreilly.com
* 68 vty 2     weak       idle                 00:00:00   freebsd.oreilly.com

  Interface      User         Mode                         Idle     Peer Address

Router1>
```

More importantly, log messages will capture the username of the individual who invoked certain high-profile commands such as configuration changes, the clearing of counters, and reloads. For example:

```
Jun 27 12:58:26: %SYS-5-CONFIG_I: Configured from console by ijbrown on vty2
 (172.25.1.1)
Jun 27 13:02:22: %CLEAR-5-COUNTERS: Clear counter on all interfaces by weak on vty2
 (172.25.1.1)
Jun 27 14:00:14: %SYS-5-RELOAD: Reload requested by kdooley on vty0
 (172.25.1.1).
```

Note that these log messages now include the username associated with each action. So, instead of just knowing that somebody changed the configuration or reloaded the router, you can see exactly who did it.

In addition, the router captures the username of the last person to modify its configuration or save the configuration to NVRAM. To see this information, use the *show running-config* command:

```
Router1#show running-config
Building configuration...

Current configuration : 4285 bytes
!
! Last configuration change at 12:58:26 EDT Fri Jun 27 2003 by ijbrown
! NVRAM config last updated at 13:01:45 EDT Fri Jun 27 2003 by kdooley
!
version 12.2
```

The *username* command also has an *autocommand* keyword, which you can use to assign an EXEC-level command to a particular username. This is useful when you want to provide limited access to a particular command while restricting access to everything else on the router. For example, you might want to set up a special username that anybody could use to run a single router command and then terminate the session:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#aaa new-model
Router1(config)#aaa authentication login default local
Router1(config)#aaa authorization exec default local
Router1(config)#username run nopassword noescape
Router1(config)#username run autocommand show ip interface brief
Router1(config)#end
Router1#
```

In this example, we defined the username *run* without a password and assigned it an *autocommand* of *show ip interface brief* . When you log in to the router with this username, the router will not prompt for a password. It just automatically executes the command and then terminates the session:

```
Freebsd% telnet Router1
Trying  172.22.1.4...
```

```
Connected to Router1.
Escape character is '^]'.

User Access Verification

Username: run
Interface                  IP-Address    OK? Method Status                Protocol
BRI0/0                     unassigned    YES NVRAM  administratively down down
Ethernet0/0                172.25.1.8    YES NVRAM  administratively down down
BRI0/0:1                   unassigned    YES unset  administratively down down
BRI0/0:2                   unassigned    YES unset  administratively down down
FastEthernet1/0            172.22.1.4    YES NVRAM  up                    up
Loopback0                  192.168.20.1  YES NVRAM  up                    up
Connection closed by foreign host.
Freebsd%
```

Note that the router issued the command and then terminated the session without providing an opportunity to issue another command.

The *noescape* keyword prevents the user from issuing an escape sequence to access the router EXEC. We strongly recommend using this keyword whenever you use autocommands.

3.1.4 See Also

Recipe 3.2 ; Recipe 3.4 ; Recipe 3.19 ; Recipe 3.22 ; Chapter 4

Top

Recipe 3.2 Encrypting Passwords

3.2.1 Problem

You want to encrypt passwords so that they do not appear in plain-text in the router configuration file.

3.2.2 Solution

To enable password encryption on a router, use the *service password-encryption* configuration command:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#enable password oreilly
Router1(config)#line vty 0 4
Router1(config-line)#password cookbook
Router1(config-line)#line con 0
Router1(config-line)#password cookbook
Router1(config-line)#line aux 0
Router1(config-line)#password cookbook
Router1(config-line)#exit
Router1(config)#service password-encryption
Router1(config)#end
Router1#
```

This command uses a weak, reversible encryption method to encipher VTY and enable passwords. See Recipe 3.5 for more details.

3.2.3 Discussion

By default, the router stores all passwords in clear-text and presents them in a human-readable format when you look at the router's configuration. The *service password-encryption* command encrypts the passwords using the Vigenere encryption algorithm. Unfortunately, as we illustrate in Recipe 3.5, the Vigenere encryption method is cryptographically weak and trivial to reverse.

However, this functionality is still quite useful to prevent nosy neighbors from viewing passwords over your shoulder. As such, encrypting your passwords is still highly recommended in spite of the known weaknesses. You should be aware of the inherent weaknesses of this encryption scheme when storing or forwarding router configuration files though. Recipe 3.4 provides a small utility to strip your router

configuration files of all passwords (encrypted or not) to keep stored and forwarded configuration files safe from prying eyes.

A configuration file with password encryption enabled looks like this:

```
Router1#show running-config
Building configuration...

Current configuration : 4385 bytes
!
! Last configuration change at 13:08:35 EDT Thu Jun 27 2002 by weak
! NVRAM config last updated at 13:01:45 EDT Thu Jun 27 2002 by kdooley
!
version 12.2
service password-encryption

!
hostname Router
!
enable password 7 06091D2445420500
!
username ijbrown password 7 045802150C2E
username kdooley password 7 070C285F4D06
!
line con 0
 password 7 0605002E474C06160E
line aux 0
 password 7 151104030F28242B23
line vty 0 4
 password 7 110A160A1C1004030F
 !
end
```

Note that the router now encrypts all passwords and no longer displays them in a human-readable format.

3.2.4 See Also

Recipe 3.3; Recipe 3.4; Recipe 3.5

Top

Recipe 3.3 Using Better Encryption Techniques

3.3.1 Problem

You want to assign a privileged password using a stronger encryption standard than Cisco's trivial default encryption.

3.3.2 Solution

To enable strong, nonreversible encryption of the privileged password, use the *enable secret* configuration command:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#enable secret ORAbooks
Router1(config)#end
Router1#
```

3.3.3 Discussion

Cisco introduced *enable secret* to improve the security of the *enable password* command. This command uses the cryptographically strong MD5 algorithm to encrypt passwords. There are no known methods for reversing this algorithm. When you configure the router with an enable secret password, it encrypts your enable password whether you have the *service password-encryption* command or not. The *service password-encryption* command has no effect on the enable secret password.

Configuring a nonreversible enable password provides greater security than the traditional enable password command. It is useful in environments that store or transfer configuration files across the network. The enable secret password takes precedence over the enable password—if you have both types of enable passwords configured, the router uses only the secret version. We highly recommend using the enable secret password on all routers.

The following command shows what the *enable secret* command looks like in the router's configuration file:

```
Router1#show running-config | include secret
enable secret 5 $1$Ahxf$4OivEQn0n0JneSObfRdSw0
Router1#
```

The following is a list of *enable secret* password restrictions:

- The password must contain between 1 and 25 alphanumeric characters (upper- or lowercase).
- Leading spaces are ignored, while intermediate and trailing spaces are permitted and recognized.
- You can use a question mark (?) in the password, but only if you precede the question mark with a Ctrl-V.

You should never use the same password for the enable password and enable secret commands. The router warns you against doing this, but will accept it:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#enable password cisco
Router1(config)#enable secret cisco
The enable secret you have chosen is the same as your enable password.
This is not recommended.  Re-enter the enable secret.

Router1(config)#end
Router1#
```

Setting the same password for both commands defeats the purpose of using the enable secret command in the first place by rendering its strong encryption useless. Avoid this problem by choosing a different password or removing the enable password altogether.

> Even strong encryption is vulnerable to dictionary and brute force attacks. To protect against such attacks, ensure that all of your passwords are difficult to guess. Always avoid using words found in the dictionary. For example, we used a common password-cracking program that took less than a minute to find the password "cookbook11."

3.3.4 See Also

Recipe 3.4; Recipe 3.22

Top

Recipe 3.4 Removing Passwords from a Router Configuration File

3.4.1 Problem

You want to remove sensitive information from a router configuration file.

3.4.2 Solution

The following Perl script removes sensitive information such as passwords and SNMP community strings from configuration files. The script takes the name of the file containing the router's configuration as its only command-line argument.

Here's some example output:

```
Freebsd% strip.pl Router1-confg

version 12.2
service password-encryption
!
hostname  Router1
!
aaa new-model
aaa authentication login default local
enable secret <removed>
enable password <removed>
!
username ijbrown password <removed>
username kdooley password <removed>
!
!Lines removed for brevity
!
!
snmp-server community <removed> RO
snmp-server community <removed> RW
!
line con 0
 password <removed>
line aux 0
 password <removed>
line vty 0 4
 password <removed>
 end
Freebsd%
```

Example 3-1 contains the Perl script.

Example 3-1. strip.pl

```
#!/usr/local/bin/perl
#
#       strip.pl   -- a script to remove sensitive information
#                     from a router configuration file.
#
#
my $configf;
undef $/;
#
$configf = shift(@ARGV);
if (open (CNFG, $configf ) ){
          $config=<CNFG>;
          close (CNFG);
          $config =~ s/password .*/password <removed>/gi;
          $config =~ s/secret .*/secret <removed>/gi;
          $config =~ s/community [^ ]+/community <removed>/gi;
          print $config;
} else {
        print STDERR "Failed to open config file \"$configf\"\n";
        }
```

3.4.3 Discussion

This script strips sensitive information from router configuration files. You can safely store or forward the resulting "stripped" configuration files to others, including vendors, partners, or colleagues. Recipe 3.5 shows how trivial the default password-encryption method is, which highlights why stripping a configuration file like this is so important.

This script should require no modifications to work in most environments. Because the script sends its output to the screen, you will have to direct the standard output into a file if you want to save a copy of the "stripped" configuration file:

```
Freebsd% strip.pl Router1-confg > /tmp/Router1-stripped
```

This example runs the script and sends the output to a file called *Router1-stripped* in the directory */*. Of course, you can direct the output of the script to any file you wish.

In earlier recipes, we mentioned that the *enable secret* password was encrypted using a strong method, MD5, and that no known method of reversing it exists. However, you may still be vulnerable to brute force attacks in which an attacker systematically encrypts likely sequences of letters, numbers, and characters in an attempt to find an encrypted match. Although these types of attacks are time-consuming, there are a number of freely available software packages that offer efficient password cracking capabilities. In short, it is better to be safe than sorry.

You can easily modify the script to strip other sensitive configuration commands (such as TACACS keys, routing keys, etc.) by simply adding more substitution lines. For instance, to strip TACACS keys, add the following line of code near the other lines that begin with `$config =~`:

```
$config =~ s/tacacs-server key .*/tacacs-server key <removed>/gi;
```

3.4.4 See Also

Recipe 3.2; Recipe 3.3; Recipe 3.5

Top

Recipe 3.5 Deciphering Cisco's Weak Password Encryption

3.5.1 Problem

You want to reverse the weak Cisco password encryption algorithm to recover forgotten passwords.

3.5.2 Solution

To recover a lost router password from a configuration file, use the following Perl script to decipher weakly encrypted passwords. This script expects to read router configuration commands via STDIN. It then prints the same commands to standard STDOUT with the passwords decrypted.

Here is an example of the program's output:

```
Freebsd% cpwcrk.pl < Router1-confg

version 12.2

service password-encryption
!
hostname Router1
!
enable secret 5 $1$4y6Q$bcGReJ3kGgmlpfr7/lT64.
enable password 7 06150E2F4A5C0817 (decrypted: sanfran)
!
username ijbrown password 7 121A0C041104 (decrypted: cisco)
username kdooley password 7 1306181D000E0B2520 (decrypted: cookbook)
!
<Lines removed for brevity>
!
line con 0
 password 7 06120A22445E1E1D (decrypted: techpwd)
line aux 0
 password 7 021201580316182 5 (decrypted: techpwd)
line vty 0 4
 password 7 070033494705151C (decrypted: oreilly)
 login
!
end
```

Example 3-2 contains the Perl script.

Example 3-2. cpwcrk.pl

```
#!/usr/local/bin/perl
#
#  cpwcrk.pl  -- a small script to crack Cisco's Type 7 password
#                encryption
#
#
$k='dsfd;kfoA,.iyewrkldJKDHSUB';
for($i=0; $i<length($k); $i++) { $ks[$i] = ord(substr($k, $i, 1)); }

while (<STDIN>) {
   if(/ord 7 [01]/) {
      chop; $w=$_; s/.* //g; $C = $_;
      printf "$w (decrypted: ";

      $o=substr($C, 0, 2);
       for ($i=0; $i < (length($C)-1)/2; $i++) { $cs[$i]=hex(substr($C,2*$i,2)); }

      for ($j=1; $j < $i; $j++) { printf("%c", $ks[$o+$j-1] ^ $cs[$j]); }
      printf ")\n";
   } else {
     printf $_;
   }
}
```

Note that this script will not decrypt *enable secret* passwords, because they are encrypted using strong MD5 encryption.

3.5.3 Discussion

This little Perl script is deliberately written to be small and fast, which, unfortunately, makes it somewhat difficult to read. Here's a brief explanation of how it works.

The first thing it does is to take the standard key string, "$k", and translate it into an array of hexadecimal numbers, "$ks", to make it easier to work with. Then it reads the input configuration file one line at a time, looking for any lines including "password 7". To make the search slightly quicker, the script looks only for the string "ord 7", followed by a space and either the number 0 or 1. It stores the encrypted password string in the variable "$C".

The first two characters in the encrypted string are used for an offset, so the script stores them in the string "$o". It then goes through the crypt string, two characters at a time, and converts the result into hexadecimal numbers, which it stores in the array "$cs".

The script then does all of the actual decryption work in a loop that simply calculates an XOR (exclusive OR) operation between the offset original key and the elements of the "$cs" array.

The encryption algorithm is nothing more than a bitwise XOR between the password string and a known key offset by a random amount. This random offset is encoded into the first two characters of the result, so it is easy to uniquely reverse the cipher.

Cisco chose to use such a simple algorithm because the router must be able to uniquely decrypt passwords for some applications, such as CHAP authentication. This is different from most password applications in which the device can take an incoming password string, encrypt it with a one-way algorithm, and compare the resulting encrypted version with the encrypted version of the locally stored password. In those applications, it is sufficient to work only with the encrypted versions of the passwords.

3.5.4 See Also

Recipe 3.2; Recipe 3.3; Recipe 3.4

Top

Recipe 3.6 Displaying Active Users

3.6.1 Problem

You want to find out who else is logged into a router.

3.6.2 Solution

To see which users are currently logged into the router and on which line, use the *show users* EXEC command:

```
Router1#show users
```

Use the keyword *all* to view all lines, including those that are inactive:

```
Router1#show users all
```

The EXEC command *who* gives the same output as the *show users* command:

```
Router1#who
```

To remotely view which users are logged into a router, use the *finger* command from your management server:

```
Freebsd% finger @Router1
```

This last command works only if the *finger* service is enabled on the router.

3.6.3 Discussion

The router provides a number of different methods to view active users. The output from all of these commands is nearly identical. Many administrators like to know which users are accessing the router for security purposes, operational reasons, or just out of curiosity.

The format of the output is as follows: the absolute line number, the VTY line number, the username, a listing of connected hosts, the inactivity timer, and the source address of the session. Note that one line of the output has an asterisk (`*`) in the left margin, indicating your current session.

The *show users* command displays the current active users and their associated line information:

```
Router1#show users
    Line       User       Host(s)              Idle       Location
    66 vty 0   ijbrown    idle                 00:56:15   freebsd.oreilly.com
    67 vty 1   kdooley    idle                 00:17:52   freebsd.oreilly.com
  * 68 vty 2   weak       idle                 00:00:00   freebsd.oreilly.com

  Interface    User       Mode                  Idle      Peer Address

Router1#
```

If you add the keyword *all* to this command, the router displays all of its lines, active and inactive:

```
Router1#show users all
    Line       User       Host(s)              Idle       Location
    0 con 0                                    00:00:00
    65 aux 0                                   00:00:00
    66 vty 0   ijbrown    idle                 00:56:24   freebsd.oreilly.com
    67 vty 1   kdooley    idle                 00:18:01   freebsd.oreilly.com
  * 68 vty 2   weak       idle                 00:00:00   freebsd.oreilly.com
    69 vty 3                                   00:00:00
    70 vty 4                                   00:00:00

  Interface    User       Mode                 Idle       Peer Address

Router1#
```

The *who* command is named after the popular Unix program, which displays active users. The router's version of *who* displays exactly the same information as the *show users* command:

```
Router1#who
    Line       User       Host(s)              Idle       Location
    66 vty 0   ijbrown    idle                 00:56:58   freebsd.oreilly.com
    67 vty 1   kdooley    idle                 00:18:36   freebsd.oreilly.com
  * 68 vty 2   weak       idle                 00:00:00   freebsd.oreilly.com

  Interface    User       Mode                 Idle       Peer Address

Router1#
```

The *finger* command is another popular Unix program that displays the active users of a remote system using a simple open IP-based protocol. The router responds to any finger request with output similar to that of the *show users* command. In the following example, we use *finger* from a Unix server to see which users are logged into a particular router:

```
Freebsd% finger @Router1
[Router1]

    Line       User       Host(s)              Idle       Location
  * 66 vty 0               idle                 00:00:00   freebsd.oreilly.com
    67 vty 1   ijbrown    idle                 00:01:48   freebsd.oreilly.com
    69 vty 3   ijbrown    idle                 00:59:04   freebsd.oreilly.com
```

```
   Interface   User         Mode                 Idle       Peer Address
Freebsd%
```

Notice that we were able to remotely extract the active user list without even logging into the router. For security purposes, we recommend that you disable the finger service to prevent illegitimate use of the protocol. For example, somebody could use this command to discover a valid username as well as a remote workstation that is allowed to log into the router. This can be a dangerous amount of information to give away freely.

You can disable the finger service on a router with the following configuration command:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#no ip finger
Router1(config)#end
Router1#
```

3.6.4 See Also

Recipe 2.6; Recipe 3.1; Recipe 3.7; Chapter 4

Top

Recipe 3.7 Sending Messages to Other Users

3.7.1 Problem

You want to send a message to another user logged into the same router.

3.7.2 Solution

To send a text message to all active users logged into a router, use the *send* EXEC command. You must have administrator privileges to use this command:

```
Router1#send *
```

To send a private message to a user logged onto a specific line, use the *send* command with the line number:

```
Router1#send 66
```

To send a private message to a user on the AUX port:

```
Router1#send aux 0
```

To send a private message to a user on the console port:

```
Router1#send console 0
```

To send a private message to a user on a specific VTY port:

```
Router1#send vty 2
```

3.7.3 Discussion

Sending messages to other users on a router is quite useful. You might want to use this ability to warn other users that you are about to reload or make changes to the router. This is a particularly valuable feature when remote users are located in different geographical areas. You can exchange messages with other users immediately without having to track down individuals via phone, pager, or cell phone.

We often use this feature while troubleshooting network problems. It is particularly useful for communicating with an onsite technician connected to the router's console, especially if you have no other means to reach them. This is a great way to coordinate everybody's efforts when there are no

telephones near the router, and cell phones won't work in an electrically noisy equipment room.

To view all of the active users on the router, use the *show users* EXEC command:

```
Router1#show users
    Line       User       Host(s)              Idle       Location
    66 vty 0   ijbrown    idle                 01:08:46   freebsd.oreilly.com
    67 vty 1   kdooley     idle                 00:05:34   freebsd.oreilly.com
  * 68 vty 2   weak       idle                 00:00:00   freebsd.oreilly.com

    Interface  User       Mode                 Idle       Peer Address

Router1#
```

In this example, we send a message to user *kdooley*, who is connected to VTY1:

```
Router1#send vty 1
Enter message, end with CTRL/Z; abort with CTRL/C:
Kev,

I need to reload this router to clear a fragmented memory problem.
Please save your work and log off, ASAP...  Thanks

IJ
^Z
Send message? [confirm]
Router1#
```

Once you submit the *send* command, the router enters a text editor mode. At this point, you can enter any text. To exit the text editor mode, type Ctrl-Z. The router then prompts you for a confirmation and, if you confirm, it sends a message that might look something like this:

```
Router1#

***
***
*** Message from tty68 to tty67:
***
Kev,

I need to reload this router to clear a fragmented memory problem.
Please save your work and log off, ASAP...

IJ
Router1#
```

3.7.4 See Also

Recipe 3.6

Top

Recipe 3.8 Changing the Number of VTYs

3.8.1 Problem

You want to increase or decrease the number of users who can simultaneously telnet to the router.

3.8.2 Solution

If you want to increase the number of VTY ports available on the router for remote access, you just need to create a reference to the additional lines in the configuration as follows:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#line vty 0 9
Router1(config-line)#end
Router1#
```

This command defines the characteristics for a range of VTY ports from 0 to 9. Since ports 0 to 4 exist by default, this has the effect of creating ports 5 through 9.

3.8.3 Discussion

By default, most Cisco routers provide five VTYs for remote access. However, the default number of VTYs is often insufficient, and increasing the number can be quite useful. This is particularly true in lab or training environments that require a large number of concurrent sessions on a particular router. In addition, organizations that disable EXEC timeouts (as shown in Recipe 3.9) often require a larger number of VTYs to prevent locking administrators out of their routers.

The router can support up to 181 virtual terminals. However, it is extremely rare to actually need more than about 20. Keep in mind that additional virtual terminals will utilize system resources, so don't go overboard. You must explicitly configure all of the new VTY lines with passwords, access classes, EXEC timeouts, transport protocols, and so forth.

To view the newly created VTY terminals, use the *show users all* EXEC command:

```
Router1#show users all
    Line       User       Host(s)              Idle       Location
   0 con 0                                    00:00:00
  65 aux 0                                    00:00:00
```

```
  66 vty 0     ijbrown    idle                  01:15:29 freebsd.oreilly.com
  67 vty 1     kdooley    idle                  00:12:17 freebsd.oreilly.com
* 68 vty 2     weak       idle                  00:00:00 freebsd.oreilly.com
  69 vty 3                                      00:00:00
  70 vty 4                                      00:00:00
  71 vty 5                                      00:00:00
  72 vty 6                                      00:00:00
  73 vty 7                                      00:00:00
  74 vty 8                                      00:00:00
  75 vty 9                                      00:00:00

  Interface      User         Mode                    Idle     Peer Address

Router1#
```

Five new VTY lines are now available on this router (ports 5 through 9).

To remove the newly created VTY lines, use the *no* version of the command:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#no line vty 5
Router1(config)#end
Router1#show users all
    Line       User       Host(s)              Idle       Location
   0 con 0                                     00:00:00
  65 aux 0                                     00:00:00
* 66 vty 0     ijbrown    idle                 00:00:00 freebsd.oreilly.com
  67 vty 1                                     00:00:00
  68 vty 2                                     00:00:00
  69 vty 3                                     00:00:00
  70 vty 4                                      00:00:00

  Interface      User         Mode                    Idle     Peer Address

Router1#
```

You cannot create or delete VTY lines out of order. Adding VTY line 20 automatically creates lines numbered from 5 to 20. Similarly, removing VTY line 5 implicitly removes all lines above line 5 (as illustrated in the previous example).

The router will not allow you to remove the original five virtual terminals. If you do attempt to delete them, the router produces the following warning message:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#no line vty 4
% Can't delete last 5 VTY lines
Router1(config)#end
Router1#
```

3.8.4 See Also

Recipe 3.9; Recipe 3.10; Recipe 3.16

Top

Recipe 3.9 Changing VTY Timeouts

3.9.1 Problem

You want to prevent your Telnet session from timing out.

3.9.2 Solution

To prevent Telnet (or SSH) sessions from timing out, use the following command:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#line vty 0 4
Router1(config-line)#exec-timeout 0 0
Router1(config-line)#end
Router1#
```

You can use this same command to simply increase the EXEC timeout to a large value, such as 4 hours:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#line vty 0 4
Router1(config-line)#exec-timeout 240 0
Router1(config-line)#end
Router1#
```

3.9.3 Discussion

By default, the router terminates an EXEC session after 10 minutes of inactivity. Administrators often find that 10 minute inactivity timers are a nuisance and dislike having to log into a router several times throughout the day. So Cisco provides a way to modify or disable the inactivity timer. It is important to note that this affects only timeouts due to inactivity. In Recipe 3.11 we discuss a way to disconnect sessions after a specified length of time, whether they are active or not.

The *exec-timeout* command has two arguments:

```
Router1(config-line)#exec-timeout 240 0
```

The first argument is the length of time in minutes, and the second argument is time in seconds. This allows you to specify a timeout period as short as 1 second, or as long as 35,791 minutes (which is over 24 days).

The first example shows how to disable the inactivity timer altogether, by setting the timeout values to zero. There are a few drawbacks to disabling the EXEC timeout that you should bear in mind. First, since the router only provides five VTYs for remote access by default, forgotten sessions can easily block available VTYs until service is completely blocked. Second, sessions that do not terminate correctly (for example, when a workstation crashes) can cause VTY sessions to remain active indefinitely.

To prevent dead sessions from needlessly occupying VTY ports, use the *service tcp-keepalives* configuration command:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#service tcp-keepalives-in
Router1(config)#end
Router1#
```

TCP keepalives will ensure that the far end is up and active. Otherwise, it will terminate the session regardless of the inactivity timer. We strongly recommend using the TCP *keepalive* command if you choose to disable the inactivity timer.

You can see your current session's inactivity timer with the *show terminal* EXEC command:

```
Router1#show terminal
Line 68, Location: "", Type: "VT100"
Length: 43 lines, Width: 95 columns
Baud rate (TX/RX) is 9600/9600
Status: PSI Enabled, Ready, Active, No Exit Banner, Automore On
Capabilities: none
Modem state: Ready
Group codes:    0
Special Chars: Escape  Hold  Stop  Start  Disconnect  Activation
                ^^x    none   -     -     none
Timeouts:      Idle EXEC     Idle Session   Modem Answer  Session  Dispatch
               never         never                       none     not set
```

The second example sets the inactivity timer to 4 hours. This tends to be safer than completely disabling the timer because it will eventually terminate all sessions. However, please check your local security policies to ensure that your inactivity timers are set within your organizational guidelines. Many organizations mandate a 15-minute inactivity timer for all types of electronic access to ensure that you do not leave authenticated sessions available to intruders. The NSA recommends an inactivity timer of no more than 5 minutes.

3.9.4 See Also

Recipe 3.11; Recipe 3.14

Top

Recipe 3.10 Restricting VTY Access by Protocol

3.10.1 Problem

You want to restrict what protocols can be used to access the router's VTY ports.

3.10.2 Solution

To restrict what protocols that you can use to access the router's VTY ports, use the *transport input* configuration command:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#line vty 0 4
Router1(config-line)#transport input telnet
Router1(config-line)#end
Router1#
```

3.10.3 Discussion

Most administrators do not realize that, by default, Cisco routers allow VTY access via protocols other than Telnet. In some instances, intruders can bypass security measures that you have in place for Telnet and access your VTYs directly. To be safe, we recommend disabling all unused protocols from accessing your VTYs. This prevents unauthorized VTY access through one of these other protocols.

Our example shows how to restrict VTY access to Telnet only. Of course, your organization may require the inclusion of other protocols, such as SSH. Recipe 3.20 discusses how to enable the SSH protocol and prevent all other forms of nonsecure access.

Table 3-1 lists the valid protocols supported by Cisco router VTYs.

Table 3-1. VTY input transport protocols

Keyword	Description
all	Enables all protocols
lat	Enables digital LAT protocol connections

Keyword	Description
mop	Enables Maintenance Operation Protocol (MOP) transport
nasi	Enables NetWare Access Servers Interface (NASI) transport
none	Disables all input protocols
pad	Enables X.3 PAD connections
rlogin	Enables the Unix rlogin protocol
ssh	Enables Secure Shell (SSHv1) protocol
telnet	Enables inbound Telnet connections
v120	Enables the V.120 protocol

Use the *show terminal* EXEC command to view the permitted protocol types for the active line. A router with the default configuration returns a long list of allowed protocols:

```
Router1#show terminal | include input
Allowed input transports are lat pad v120 lapb-ta telnet rlogin ssh.
Router1#
```

After we restrict the VTY access to Telnet only, the output looks like this:

```
Router1#show terminal | include input
Allowed input transports are telnet.
Router1#
```

3.10.4 See Also

Recipe 3.9; Recipe 3.14; Recipe 3.16

Top

Recipe 3.11 Enabling Absolute Timeouts on VTY Lines

3.11.1 Problem

You want to enable absolute timeouts on your VTY lines.

3.11.2 Solution

To enable absolute VTY timeouts, use the following set of configuration commands:

```
Router1# configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#line vty 0 4
Router1(config-line)#absolute-timeout 5
Router1(config-line)#logout-warning 30
Router1(config-line)#end
Router1#
```

3.11.3 Discussion

To prevent users from indefinitely tying up valuable router VTY lines, you can implement absolute timers. Absolute timers differ from the inactivity timers discussed in Recipe 3.9 because they will terminate a session whether it is active or not. Although absolute timers are rarely enabled, there are times when they can be quite useful. For example, in classroom and lab situations, the absolute timeout can help to ensure that nobody uses more than their fair share of login time.

The prospect of having a session terminated in the middle of troubleshooting a problem in a production network is not appealing to most administrators. So, if you do choose to implement an absolute timer, we recommend setting the timer to a reasonable amount of time (no less than 10 minutes). In addition, you should enable a logout warning to ensure that the user has plenty of notice to save their work. The following is an example of a logout warning banner:

```
Router1>
*
*
* Line timeout expired
*
*
Router1>Connection closed by foreign host.
Freebsd%
```

Note that the argument for the *absolute-timeout* command is a time value in minutes whereas the *logout-warning* command uses seconds. In the example, we set the absolute timeout to 5 minutes and the warning message to 30 seconds. A 30-second warning may be too aggressive in a production environment.

3.11.4 See Also

Recipe 3.9

Top

Recipe 3.12 Implementing Banners

3.12.1 Problem

You want to implement a banner message to display a security warning.

3.12.2 Solution

The following commands configure various types of banners on a router:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#banner exec  # This is an exec banner #
Router1(config)#banner login # This is a login banner #
Router1(config)#banner motd  $ This is a motd banner  $
Router1(config)#end
Router1#
```

Note that the router will accept almost any delimiter character as long as the start and end delimiters are identical. These delimiters allow you to make your banner message several lines long. Our first two examples use the pound symbol (#), while the last example uses the dollar sign ($) as a delimiter. Be careful—if the delimiter character appears within the banner message itself, the router will only accept part of the message.

3.12.3 Discussion

Cisco routers support three main types of banners and display them in a strict order. The message of the day (motd) is followed by the login banner before the login prompt, and the router prints the EXEC banner after a successful authentication:

```
Freebsd% telnet Router1
Trying 172.22.1.4...
Connected to Router1.
Escape character is '^]'.
 This is a motd banner
 This is a login banner

User Access Verification

Username: ijbrown
```

```
Password: <xxxxxxxxx>
 This is an exec banner
Router1>
```

Login banners are mainly used to display a warning message for security purposes; we will discuss this in a moment. The *motd* banner derives from the Unix banner bearing the same name. The *motd* banner is of little use in production environments and is rarely required. The EXEC banner, on the other hand, is useful for displaying administrator messages because it is only presented to authenticated users (much like the Unix *motd* banner).

Banners are an important and often overlooked part of a good security policy. Although a banner alone will not repel the crafty hacker, it provides a certain level of legal protection. In fact, a well-designed warning message may indeed repel a would-be hacker: the mere threat of legal action can be a wonderful deterrent. If unauthorized users suspect that your organization is serious about legal action, they are less likely to target your devices. We highly recommend implementing login banners on all production routers.

A good login banner should meet the following objectives:

- Notify people who attempt to access the router that unauthorized use is prohibited and that only authorized users with official business are permitted access.
- Tell users that they should have no expectation of privacy, since all activities may be monitored and/or recorded without further notification.
- Remind users that unauthorized access is unlawful and that recorded logs may be used in civil and/or legal action.
- Most importantly, the banner should never surrender sensitive information about the router, your organization, or any other piece of information that could aid a hacker.

Login banners can simplify the prosecution of hackers who unlawfully access your system by explicitly notifying unauthorized users that their actions are indeed unauthorized. Think of the banner as the electronic equivalent of a sign saying, "trespassers will be prosecuted." Without this sign, somebody could theoretically claim that they didn't know it was a private system. It may not hold up in court, but why take the risk? Laws governing legal notification vary significantly between jurisdictions and situational purpose. We recommend that you clear all proposed banners with your legal department before implementation.

The following banner message show a particularly well-written legal notice that meets all of requirements mentioned earlier. The FBI's Atlanta computer crime squad provided this example banner. Again, please check with your local authorities before creating a warning banner to ensure that it meets your local legal requirements:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#banner login #
Enter TEXT message.  End with the character '#'.

+---------------------------------------------------------------------+
|                              WARNING                                |
|                              -------                                |
| This system is solely for the use of authorized users for official  |
| purposes.  You have no expectation of privacy in its use and to     |
| ensure that the system is functioning properly, individuals using   |
| this computer system are subject to having all of their activities  |
| monitored and recorded by system personnel. Use of this system      |
| evidences an express consent to such monitoring and agreement that  |
| if such monitoring reveals evidence of possible abuse or criminal   |
| activity, system personnel may provide the results of such          |
| monitoring to appropriate officials.                                |
+---------------------------------------------------------------------+
#
Router1(config)#end
Router1#
```

Starting with Version 12.0(3)T of IOS, Cisco routers began to support banner token functionality. Tokens are variables (listed in Table 3-2) that are embedded within a banner message and serve as substitutes for things such as hostnames and domain names.

Table 3-2. Supported banner tokens list

Token name	Substituted information
$(hostname)	Displays the router's hostname
$(domain)	Displays the configured domain name
$(line)	Displays the active line number
$(line-desc)	Displays a description of the active line

Tokens allow you to distribute a single banner message throughout your network by using variable substitution to make it look slightly different on each device. This ensures that any local differences in the information are always accurate. The banner message can dynamically adapt to changes in hostname or line number, for instance.

Although all banner types support tokens, we recommend using them only in EXEC banners. Since tokens surrender information about the router, it is inappropriate to use them within login or *motd* banners, which are visible before the user supplies a valid username or password. EXEC banners, on the other hand, are visible only to authenticated users. The following example shows how to configure

an EXEC banner with tokens:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#banner exec #
Enter TEXT message.  End with the character '#'.
Welcome, you have connected to router $(hostname).$(domain):
on line $(line) ($(line-desc)).
#
Router1(config)#line vty 0 4
Router1(config-line)#location 999 Queen Street West
Router1(config)#end
Router1#exit
Connection closed by foreign host.
Freebsd% telnet Router1
Trying 172.25.1.7...
Connected to Router1.
Escape character is '^]'.


User Access Verification

Password: <vtypassword>
Welcome, you have connected to router Router1.oreilly.com:
on line 5 (999 Queen Street West).

Router1>
```

Note that the router substitutes the appropriate router information where the tokens were. For example, it replaces the hostname token, `$(hostname)`, with the hostname, `Router1`. The domain token, `$(domain)`, is derived from the *ip domain-name* command. The line token, `$(line)`, is replaced with the active line number. Finally, the line description token, `$(line-desc)`, is derived from the information configured with the *location* command for this line.

3.12.4 See Also

Recipe 3.13

Top

Recipe 3.13 Disabling Banners on a Port

3.13.1 Problem

You want to disable the banner on a particular port to prevent it from confusing an attached device such as a modem.

3.13.2 Solution

To disable banners on particular lines, use the following commands:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#line aux 0
Router1(config-line)#no motd-banner
Router1(config-line)#no exec-banner
Router1(config-line)#end
Router1#
```

3.13.3 Discussion

By default, the router displays the configured banner messages on all of its lines. However, there are circumstances when these banner messages can confuse directly attached devices. For instance, modems connected to terminal server or AUX lines can react erratically to banner messages (particularly motd and login banners). In these situations, you will need to disable the banner on the associated line.

Note that you cannot disable the login banner on a line-by-line basis. So, if you find that you need to do this, you should use the *motd* banner instead of the login banner to display the warning message. The *motd* and the login banners are nearly identical, so this shouldn't cause any problems.

3.13.4 See Also

Recipe 3.12

Top

Recipe 3.14 Disabling Router Lines

3.14.1 Problem

You want to disable your router's AUX port to help prevent unauthorized access.

3.14.2 Solution

To completely disable access via the router's AUX port, use the following set of commands:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#line aux 0
Router1(config-line)#transport input none
Router1(config-line)#no exec
Router1(config-line)#exec-timeout 0 1
Router1(config-line)#no password
Router1(config-line)#end
Router1#
```

You can also disable access to the router through the VTY lines:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#access-list 98 deny any log
Router1(config)#line vty 0 4
Router1(config-line)#transport input none
Router1(config-line)#exec-timeout 0 1
Router1(config-line)#no exec
Router1(config-line)#access-class 98 in
Router1(config-line)#end
Router1#
```

3.14.3 Discussion

It is extremely important to secure access to your routers. The most effective way to secure router ports is to simply disable them if they aren't needed. For instance, network administrators rarely use the router's AUX port, so they should consider disabling it. If your routers are physically close to the administrators and remote access is not necessary, you might even want to disable the VTY ports to provide greater security.

Our first example shows a somewhat paranoid method of disabling a router's AUX port. We say paranoid because it involves using several different techniques, each of which should be sufficient alone, strung together for added protection. The *transport input none* command prevents reverse Telnet access from the network (see Recipe 3.10). The *no exec* command prevents the AUX port from running an EXEC session. Without an EXEC process, no response is given when you connect a terminal to the port. The *exec-timeout* command sets the EXEC timeout to one second (see Recipe 3.9). This effectively limits the ability to submit commands, just in case an attacker somehow manages to start an EXEC process. Finally, the *no password* clears the line password, so there is no way to authenticate if somebody is able to connect.

The way we disable the VTY ports is also paranoid but effective. The example we gave illustrates several different methods to disable connectivity. In reality, any one of these methods would work, but we recommend including at least one secondary method as a safety measure. It is important to note that these extra precautions don't cost anything. They don't increase the router's CPU load or memory consumption appreciably. So, for the extra level of safety, you might as well string together several methods as we have shown.

To disable the VTY interfaces we first set the *transport input* to none (see Recipe 3.10) to disable all inbound transport methods. We also set the *exec-timeout* to one second (see Recipe 3.9). We then disable the EXEC process on this line, rendering the VTY port useless. Finally, we implement an input *access-class* that prevents all IP addresses from accessing the VTY ports (see Recipe 3.16).

If you try to connect to a VTY line that is disabled (as shown in this recipe), the router refuses the connection:

```
Freebsd% telnet Router1
Trying 172.22.1.4...
telnet: connect to address 172.22.1.4: Connection refused
telnet: Unable to connect to remote host
Freebsd%
```

The console port is the most important access line your router has. It is the last place you can still connect to when everything else goes wrong, so we do not recommend disabling it. However, we do recommend increasing the security of your console port to provide maximum protection. By default, the console port will not prompt for a login or password. This means that unauthorized users can easily gain access to your router's EXEC without effort. Of course, the router is always vulnerable to console access via the password recovery process, which highlights why physical security is important. You should always house routers in restricted rooms or closets to ensure physical security.

To increase the security of a router's console port, use the following set of configuration commands:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#line con 0
Router1(config-line)#exec-timeout 5 0
```

```
Router1(config-line)#password ora22bkz
Router1(config-line)#login
Router1(config-line)#end
Router1#
```

This example enables a login prompt by configuring the *login* and *password* commands, and reduces the *exec-timeout* to five minutes to ensure that console sessions will disconnect themselves if somebody walks away from the terminal. Note however that login and password commands become unnecessary if you configure your router to use local authentication or AAA. The stronger authentication method overrides the configurations of all of the lines, including those of the console port.

3.14.4 See Also

Recipe 3.1; Recipe 3.9; Recipe 3.10; Recipe 3.16; Chapter 4

Top

Recipe 3.15 Reserving a VTY Port for Administrative Access

3.15.1 Problem

You want to prevent other people from using up all of your VTY lines, effectively locking you out of the router.

3.15.2 Solution

You can ensure that at least one VTY port is available to you for access at all times with the following commands:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#access-list 9 permit 172.25.1.1
Router1(config)#line vty 4
Router1(config-line)#access-class 9 in
Router1(config-line)#end
Router1#
```

3.15.3 Discussion

Receiving the dreaded "Connection refused" message from one of your routers can be quite distressing, particularly if you're trying to troubleshoot a serious problem. Generally, it means that other sessions have control of all of your router's limited number of VTY lines. However, it can also mean that someone has launched a Denial of Service (DoS) attack. DoS attacks against router VTYs are simple to launch. Just sitting at a login prompt is enough to tie up a VTY line. This means that you don't need a username or a password to use up all of the VTY lines, thus locking out all of the legitimate administrators.

Whether the lockout is caused by legitimate sessions or not, this is what it looks like:

```
Freebsd% telnet Router1
Trying 172.22.1.4...
telnet: connect to address 172.22.1.4: Connection refused
telnet: Unable to connect to remote host
Freebsd%
```

You can implement a safeguard to ensure that this never happens. Enabling a restrictive access class on the last accessible VTY ensures that the administrator will retain access at all times. The key is to

ensure that your access list is as restrictive as possible (i.e., an administrator's IP address).To view the VTY access statistics, use the *show line* command:

```
Router1#show line vty 0 4
   Tty Typ     Tx/Rx    A Modem  Roty AccO AccI   Uses   Noise  Overruns   Int
*   66 VTY               -    -      -    -    -     10       0     0/0       -
*   67 VTY               -    -      -    -    -     10       0     0/0       -
*   68 VTY               -    -      -    -    -      2       0     0/0       -
*   69 VTY               -    -      -    -    -      1       0     0/0       -
*   70 VTY               -    -      -    -    9      1       0     0/0       -

Router1#
```

Note that access class 9 was assigned to the last VTY session (the "AccI" column) and has been accessed only once ("Uses").

3.15.4 See Also

Recipe 3.16; Recipe 3.17

Top

Recipe 3.16 Restricting Inbound Telnet Access

3.16.1 Problem

You want to restrict Telnet access to the router to allow only particular workstations.

3.16.2 Solution

You can restrict which IP addresses can access the router:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#access-list 99 permit  172.25.1.0 0.0.0.255
Router1(config)#access-list 99 deny any log
Router1(config)#line vty 0 4
Router1(config-line)#access-class 99 in
Router1(config-line)#end
Router1#
```

This example uses a standard *access-list* command. You can also use extended access lists in an *access-class* statement. But, because you already know the TCP port numbers, as well as the destination IP addresses, extended access lists don't provide much extra functionality.

3.16.3 Discussion

Telnet is an inherently insecure protocol because it sends passwords over the network in clear-text. We highly recommend using *access-class* statements to help ensure that only authorized users can access router VTYs. These *access-class* statements do not secure the Telnet protocol itself, but they will help to prevent unauthorized users from receiving a router login prompt. Even if someone manages to sniff your router passwords, this will make them virtually useless.

For increased security, limit the permitted hosts to a few network management servers. This will force legitimate users to follow a two-stage authentication process to access your routers. They will need to authenticate their session on a central device such as the network management server before they can log into the router. The logic is that it is much easier to secure a single server than a dozen workstations.

This feature provides a similar functionality to the Unix *TCPWrapper* tool set, which can restrict

daemon access to a limited number of IP addresses. And, just like *TCPWrapper*, we can log the IP addresses of refused users by using the keyword *log* in the access list definition. This creates a log message for every unauthorized Telnet attempt, such as the following:

```
Router1#show logging | include list 99
Jun 27 14:14:25: %SEC-6-IPACCESSLOGS: list 99 denied 172.22.1.3 1 packet
Router1#
```

In the example, we have added an explicit *deny any* command to allow the router to count refused sessions:

```
Router1#show access-lists 99
Standard IP access list 99
    permit 172.25.1.0, wildcard bits 0.0.0.255 (4 matches)
    deny   any log (1 match)
Router1#
```

This command shows you the running total of permitted and refused Telnet sessions. In this example, the access list has denied a single Telnet session from accessing a router VTY. A large number of access attempts might indicate that someone is trying to access your routers without authorization. The log messages capture the IP source address of each denied attempt, making it easy to investigate.

3.16.4 See Also

Recipe 3.15; Recipe 3.17

Top

Recipe 3.17 Logging Telnet Access

3.17.1 Problem

You want to log every Telnet session to the router.

3.17.2 Solution

To log every Telnet session to the router, use the followings set of commands:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#access-list 90 permit any log
Router1(config)#line vty 0 4
Router1(config-line)#access-class 90 in
Router1(config-line)#end
Router1#
```

3.17.3 Discussion

Keeping detailed log records of every Telnet session your router accepts can be useful for security purposes. Configuring an access class ACL to log every session causes the router to capture which IP source addresses attempt to access the Telnet port. Note, however, that this method captures both successful and unsuccessful Telnet attempts, which is an invaluable capability.

Of course, you can combine this functionality with the other access classes that we discussed in Recipe 3.15 and Recipe 3.16. This recipe doesn't introduce any new features; it's just a different way to use the same commands.

To view all captured Telnet attempts, use the following EXEC command:

```
Router1# show logging | include list 90
Jun 27 14:44:45: %SEC-6-IPACCESSLOGS: list 90 permitted 172.25.1.1 1 packet
Router1#
```

Note that the logged messages will always show "permitted," even if the session authentication was unsuccessful.

3.17.4 See Also

Recipe 3.15; Recipe 3.16

Top

Recipe 3.18 Setting the Source Address for Telnet

3.18.1 Problem

You want to force your router to use a particular IP source address when making outbound Telnet connections.

3.18.2 Solution

To configure a single common IP source address for all outbound Telnet sessions, use the following configuration command:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#ip telnet source-interface loopback0
Router1(config)#end
Router1#
```

You can also set the IP source address for individual outbound Telnet sessions on the command line:

```
Router1#telnet 172.25.1.5 /source-interface loopback0
```

3.18.3 Discussion

By default, the router uses the IP address of the closest interface to the destination as the source address when it makes an outbound Telnet session. However, network administrators frequently want to use an address other than the one closest to the destination. For instance, access lists or route filters may block packets with the default source address. Selecting a particular source IP address for a Telnet session can also help in troubleshooting network problems.

Cisco provides two methods for setting the source IP address of Telnet sessions: global and session-specific. The global configuration example forces all Telnet sessions to use the IP address of the configured interface as the source address. However, if the configured interface has no IP address, or is operationally down, the router will resort to its default behavior of using the closest interface IP address.

The session-specific method allows the administrator to manually select an appropriate IP source address on a per-session basis. Again, if the configured interface has no IP address or is operationally down, the router will resort to its default behavior. Note that the individual method of setting the Telnet

source address supersedes the global setting.

Top

Recipe 3.19 Automating the Login Sequence

3.19.1 Problem

You want to automate the process of logging into a router, so you don't have to type usernames, passwords, and common commands.

3.19.2 Solution

The following script automates the process of logging into the router using a scripting language called Expect. Expect can be used to automate interactive sessions (see Appendix A for more details). This script takes a router name or IP address as a command-line argument. It then performs an automated login sequence before returning the session back to you for a normal interactive session.

Here's an example of the output:

```
Freebsd% tel Router1
spawn telnet Router1
Trying 172.25.1.5...
Connected to Router1.
Escape character is '^]'.

User Access Verification

Username: ijbrown
Password:

Router1>
Router1 - vty login ok
enable
Password:
Router1#
Router1 - enable login ok

Router1#term mon
Router1#
```

Example 3-3 contains the Expect code.

Example 3-3. tel

```
#!/usr/local/bin/expect
```

```
#
#           tel -- a script to perform automated login onto a Cisco
#                  router using either a hostname or IP address.
#
#
# Set behaviour
set userid ijbrown
set vtypasswd oreilly
set enablepwd cookbook
#
#
set timeout 10
set rtr [lindex $argv 0]
spawn telnet $rtr
expect {
         {Username}   { send "$userid\r"
                        expect {
                                  {*Password*} { send "$vtypasswd\r" }
                               }
                      }
         {telnet>}    { send_user "$rtr - telnet failed\n"
                        exit
                      }
         {Password}   { send "$vtypasswd\r" }
       }

expect {
         {Password}   { send_user "\n$rtr - vty login failed\n"
                        exit
                      }
         {Username}   { send_user "\n$rtr - vty login failed\n"
                        exit
                      }
         {>}          { send_user "\n$rtr - vty login ok\n" }
       }

 send "enable\r"
 expect "Password"
 send "$enablepwd\r"
#
 expect {
          {*#}        { send_user "\n$rtr - enable login ok\n" }

          {*>}        { send_user "\n$rtr - enable login failed\n"
                        exit
                      }
          {Password}  { send_user "\n$rtr - enable login failed\n"
                        exit
                      }
        }
#
send "\r"
```

```
expect "*#*"
send "term mon\r"
#
interact
```

3.19.3 Discussion

This script is intended to save you time when you have to repeatedly log into routers. The *tel* script connects to the VTY and sends the login sequence before returning the session back to you. The script can login to routers that use local usernames, AAA authentication, or the default VTY/enable passwords. You can also use it to submit router commands before returning control to the end user. Since the script can respond immediately to the various router prompts, the entire login sequence is much faster than what a human can type.

This script also notifies the user when it experiences problems in the login sequence and it displays the entire sequence so that you can follow its progress on the screen. If the script experiences a problem, it will usually terminate with an appropriate error message, if possible. It also includes a global timeout variable to ensure that problems do not hang the user session. The default global timeout is 10 seconds.

This script requires Expect to be on the server and located in the */usr/local/bin* directory. You will also need to set a few variables. First, the *userid* variable must be set to your router username, which is either the locally administered username or your AAA username. If your router does not prompt for usernames, the script ignores this variable. Second, the variable *vtypasswd* must be set to the password associated with your username. If your router is not configured to use usernames, use the VTY password. Third, the variable *enablepwd* must be set to the router's enable password.

You should store this script in your home directory with read, write, and execute privileges restricted to yourself. This ensures that unauthorized users cannot view your ID and password (which are stored in clear-text) or use the script to log into a device using your credentials:

```
Freebsd% chmod 700 tel
```

> Many corporate security organizations frown on storing unencrypted passwords in flat files. Check your security guidelines before using this script.

The final step in the script login sequence is to submit useful commands before returning the session back to the user. This time-saving step automatically submits commands that you use regularly. By default, the script will send the *terminal monitor* command before terminating; however, you can easily add other commands. You can also modify the script to send a standard set of commands and then exit from the router without needing to turn over control.

The *tel* script has proven to be an invaluable tool during the writing of this book. We have used it literally thousands of times to save countless keystrokes in the process. Think of it as a preventative measure to avoid carpal tunnel syndrome.

3.19.4 See Also

Recipe 3.1; Recipe 3.3; Chapter 4

Top

Recipe 3.20 Using SSH for Secure Access

3.20.1 Problem

You want to use SSH to give more secure encrypted remote access to your router.

3.20.2 Solution

You can configure your router to run an SSH Version 1 server for VTY access:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#hostname Router1
Router1(config)#ip domain-name oreilly.com
Router1(config)#crypto key generate rsa
The name for the keys will be: Router1.oreilly.com
Choose the size of the key modulus in the range of 360 to 2048 for your
  General Purpose Keys. Choosing a key modulus greater than 512 may take
  a few minutes.

How many bits in the modulus [512]: 1024
Generating RSA keys ...
[OK]

Router1(config)#
Jun 27 15:04:15: %SSH-5-ENABLED: SSH 1.5 has been enabled
Router1(config)#ip ssh time-out 120
Router1(config)#ip ssh authentication-retries 4
Router1(config)#end
Router1#
```

SSH became available in Cisco's IOS starting with Release 12.1(1)T. However, only versions of IOS that support IPSec (DES or 3DES) encryption include SSH support. Note that there are severe restrictions on exporting any software that includes 3DES outside of the U.S. and Canada.

3.20.3 Discussion

SSH provides a secure method of communication between network entities by means of transparent encryption. It is a protocol that encrypts all traffic, including passwords, between a client and a server. This makes it an excellent replacement for the Telnet and Rlogin protocols. Cisco's IOS currently

supports only a subset of the standard SSH tools. In particular, Cisco routers do not support the newer SSH Version 2, which includes a number of important enhancements.

The main reason to consider replacing Telnet with SSH is security. The entire Telnet session, including passwords, is transmitted in clear-text. Anybody using a protocol analyzer between the Telnet client and server can easily see all of the data sent by both ends of the conversation—including usernames and passwords. SSH, on the other hand, uses strong encryption algorithms to ensure that the entire session is unintelligible to anybody except for the intended party. This allows for secure communication through the Internet or any other public network.

The transparent encryption scheme used by SSH ensures that, except for initial configuration, SSH behaves similarly to Telnet.

Configuring SSH requires the following steps:

- Ensure that your router is running IOS Version 12.1(1)T or higher.
- Ensure that your IOS version contains the IPSec feature set (DES or 3DES). Although 3DES is preferred, IOS versions containing this feature are export controlled outside of the U.S. and Canada.
- Configure an authentication method that supports usernames and passwords, such as local authentication or AAA. SSH does not support the default VTY password encryption method. See Recipe 3.1 for information on local authentication and Chapter 4 for more information on AAA.
- Set the router's hostname to something other than the default "Router."
- Configure the *ip domain-name* on your router to match your organization's domain name.
- Finally, generate the SSH host keys using the *crypto key generate rsa* configuration command. The router can accept a key length between 360 and 2048 bits. Larger keys provide greater security, but negatively affect performance. We don't recommend using a key that is shorter than 1024 bits. Creating keys requires a large number of CPU cycles, usually a few minutes of 100% CPU utilization (depending on the router type and the key length). Once created, keys are stored in NVRAM and are inaccessible. You can delete a set of keys with the *crypto key zeroize rsa* configuration command.

Generating a set of SSH keys automatically enables the SSH protocol. As soon as you have created the keys, the router can start accepting SSH sessions. The first time you attempt to access an SSH-enabled device, your SSH client software will prompt you to store the device host key. This prevents other devices from masquerading as a legitimate device:

```
Freebsd% ssh -l ijbrown Router1
The authenticity of host 'Router1 (172.25.1.5)' can't be established.
RSA1 key fingerprint is 7a:97:99:2a:ef:08:40:fb:c3:dd:c4:8c:29:fc:2f:4d.
```

```
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'Router1' (RSA1) to the list of known hosts.
ijbrown@Router's password: xxxxxxxxxx

Router1>exit
Connection to Router1 closed.
```

SSH passes the current username to the SSH server, which in turn prompts for the password of the current user. However, with the Unix version of SSH, you can override this behavior by specifying the *-l* option, followed by an alternate username. In the previous example, we explicitly specified a particular username (*ijbrown*). The default behavior looks like this:

```
Freebsd% ssh Router1
ijbrown@Router1's password: xxxxxxxxx

Router1>
```

Because we don't specify a username in this example, the router assumes that it should use the current Unix username, *ijbrown*.

If you decide to use SSH as your transport protocol for administrative access to your routers, we recommend that you disable all other forms of VTY access by using the transport input configuration command. Running insecure protocols defeats the purpose of implementing SSH in the first place. For more information on disabling transport protocols on virtual terminals, see Recipe 3.10. The following example illustrates how to disable all inbound protocols except SSH:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#line vty 0 4
Router1(config-line)#transport input ssh
Router1(config-line)#end
Router1#
```

Starting with Version 12.1(3)T, Cisco's IOS began to support SSH client functionality as well. SSH clients allow you to access other SSH servers, including SSH-enabled routers. In the following example, we initiate an SSH session from our router to an SSH-enabled Unix server:

```
Router1#ssh -l ijbrown server
Trying server.oreilly.com (172.25.1.3)... Open

Password: xxxxxxxxxxx
FreeBSD 4.6-STABLE (IJB)

Welcome to FreeBSD!

You have new mail.
Freebsd%
```

Many SSH clients and servers are readily available for most popular operating systems. There are also

several free SSH packages available on the Internet, including OpenSSH and PuTTY (see Appendix A for more details).

The *show ssh* EXEC command displays the active SSH sessions and their attributes, such as VTY number, SSH version, encryption type, session state, and username:

```
Router1#show ssh
Connection      Version Encryption      State                   Username
 0              1.5     3DES            Session started         ijbrown
 3              1.5     3DES            Session started         morewood
```

The command *show ip ssh* displays the SSH server configuration status, including the SSH version, authentication timeout, and number of retries:

```
Router1#show ip ssh
SSH Enabled - version 1.5
Authentication timeout: 120 secs; Authentication retries: 4
Router1#
```

3.20.4 See Also

Recipe 3.1; Chapter 4

Top

Recipe 3.21 Changing the Privilege Level of IOS Commands

3.21.1 Problem

You want to change the privilege level of specific IOS commands.

3.21.2 Solution

To reduce the privilege level of an enable command from 15 to 1, use the following command:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#privilege exec level 1  show startup-config
Router1(config)#end
Router1#
```

You can also increase the privilege level of a level 1 command:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#privilege exec level 15  show ip route
Router1(config)#privilege exec level 1  show ip
Router1(config)#privilege exec level 1  show
Router1(config)#end
Router1#
```

Note that raising the privilege level of the *show ip route* command also increased the level of the *show ip* set of commands and all of the other *show* commands. In this example, we lowered the *show ip* and *show* commands back to privilege 1 to ensure that all of the other *show* commands operated normally.

3.21.3 Discussion

Cisco routers support 16 privilege levels ranging from 0 to 15. By default, Cisco assigns commands to only three of these privilege levels: zero, user, and enable. There are five commands with privilege level zero: disable, enable, exit, help, and logout. The user level (privilege level 1) has a wide variety of commands available that cannot alter the router's configuration. Enable mode (privilege level 15), by contrast, allows complete access to all router commands.

In practical terms, only levels 1 and 15 are normally used. When you first access the router using your VTY password (or local authentication), the router assigns privilege level 1 to your session. In order to

access privilege level 15, you must use the *enable* EXEC command:

```
Router1>show privilege
Current privilege level is 1
Router1>enable 15
Password:
Router1#show privilege
Current privilege level is 15
Router1#
```

You can specify any valid privilege level with the *enable* command, but the default is level 15. You can also reduce the privilege level of your current session with the *disable* command:

```
Router1#show privilege
Current privilege level is 15
Router1#disable 1
Router1>show privilege
Current privilege level is 1
Router1>
```

The *disable* command will default to privilege level 1 if you don't specify a target privilege level.

By default, Cisco assigns a subset of commands to privilege 1 and the full set of commands to privilege 15. However, the default commands for each privilege level may not be appropriate for your organization. Many organizations find it useful to modify the default command privileges.

The first example in this recipe shows how to change the privilege level of the *show start-config* from its default privilege value of 15, giving it a new value of 1. This allows normal unprivileged users to see the router's startup configuration without having to give them full enable access in the process. People usually use this feature to reduce the privilege of certain key commands. By allowing normal users to access the few commands that they need, this feature allows you to keep tighter restrictions on the commands that change the router's configuration.

Although you can change the privilege mode of any router command, the *show running-config* command does not function correctly at levels below 15:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#privilege exec level 1 show running-config
Router1(config)#end
Router1#disable
Router1>show running-config
Building configuration...

Current configuration : 85 bytes
!
! Last configuration change at 22:02:36 EDT Sun Jul 7 2002 by ijbrown
!
!
```

```
!
!
end

Router1>
```

Although the router permits the command to execute, the output is unusable.

The second recipe example shows how to increase the privilege level of a command (*show ip route*) from 1 to 15. This will prevent user-level staff from viewing the routing table. The NSA guide to Cisco security recommends that administrators increase the privilege level of the *connect*, *rlogin*, *telnet*, *show ip access-lists*, *show access-lists*, and *show logging* commands from 1 to 15:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#privilege exec level 15 connect
Router1(config)#privilege exec level 15 rlogin
Router1(config)#privilege exec level 15 telnet
Router1(config)#privilege exec level 15 show ip access-lists
Router1(config)#privilege exec level 15 show access-lists
Router1(config)#privilege exec level 15 show logging
Router1(config)#privilege exec level 1 show
Router1(config)#privilege exec level 1 show ip
Router1(config)#end
Router1#
```

Note that changing the privilege level of the *show ip route* command from 1 to 15 also increases the privilege level of all *show ip* and *show* commands to 15. So, in this example, we have explicitly reduced the privilege of these commands back to level 1 so that we don't lose access to all of the other *show* commands.

3.21.4 See Also

Recipe 3.22; Recipe 3.23

Top

Recipe 3.22 Defining Per-User Privileges

3.22.1 Problem

You want to set different privilege levels for different users.

3.22.2 Solution

To assign a particular privilege level to a user, use the following set of commands:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#aaa new-model
Router1(config)#aaa authentication login default local
Router1(config)#aaa authorization exec default local
Router1(config)#username slowell privilege 10 password  maceng#1
Router1(config)#privilege exec level 10  show ip route
Router1(config)#privilege exec level 1  show ip
Router1(config)#privilege exec level 1  show
Router1(config)#end
Router1#
```

You can also create several global privilege levels that any user can access with the appropriate password:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#enable secret level 10  lvl10passwd
Router1(config)#privilege exec level 10  show ip route
Router1(config)#privilege exec level 1  show ip
Router1(config)#privilege exec level 1  show
Router1(config)#end
Router1#
```

3.22.3 Discussion

Sometimes having two privilege level groups doesn't provide fine enough granularity. For example, you might want to have three levels of administrators: user-level staff, mid-level staff, and high-level engineers. You don't want user-level staff members to see the router's routing table; the mid-level staff should be able to see the routing table, but not make configuration changes; and the highest-level engineers need to have access to everything.

You could set up these three groups by using either method shown in the recipe example. For example, you could create user accounts for the staff members and assign the appropriate privilege level to each user or group of users. Or you could create user accounts for all of the users, and then define a series of different global enable levels. Either approach would work.

Our first example uses the *username* command, discussed in Recipe 3.1, to assign a particular privilege level to a username. We have assigned user *slowell* the privilege level 10 and increased the privilege level of the command *show ip route* to 10. Without the*aaa authorization* command, you cannot change the default privilege level. Essentially, we have created a new privilege level, 10, and assigned it a single command. It will also inherit the commands from all of the lower the privilege levels:

```
Freebsd% telnet Router1
Trying 172.22.1.4...
Connected to Router1.
Escape character is '^]'.


User Access Verification

Username: slowell
Password: <maneng#1>
Router1#show privilege
Current privilege level is 10
Router1#show ip route
Codes: C - connected, S - static, I - IGRP, R - RIP, M - mobile, B - BGP
       D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
       E1 - OSPF external type 1, E2 - OSPF external type 2, E - EGP
       i - IS-IS, L1 - IS-IS level-1, L2 - IS-IS level-2, ia - IS-IS inter area
       * - candidate default, U - per-user static route, o - ODR
       P - periodic downloaded static route

Gateway of last resort is 172.22.1.3 to network 0.0.0.0

     172.16.0.0/24 is subnetted, 1 subnets

C       172.22.1.0 is directly connected, FastEthernet1/0
O*E1 0.0.0.0/0 [110/3] via 172.22.1.3, 00:15:56, FastEthernet1/0
Router1#disable 1
Router1>show ip route
                  ^
% Invalid input detected at '^' marker.
```

Notice that when this user logs in, he automatically gets the increased privilege level without having to issue an *enable* command. He then executes the *show ip route* command, which works normally because we have assigned it to level 10. If he then reduces his level to 1 and tries the same command again, it won't work.

You could assign a username to privilege level 15 (enable level), but we do not recommend doing this.

The extra layer of password protection and the strong encryption used by the *enable secret* command outweighs the convenience of assigning a user privilege level 15.

The second example defines a new privilege level using the *enable secret* command. You can also use the *enable password* command to define per-level usernames, but the *enable secret* command gives much better encryption, as we showed in Recipe 3.5.

The second method has two distinct advantages over the first example. First, the *enable secret* command uses strong MD5 encryption to store its passwords in the configuration. Second, it ensures that the new privilege level is available to all user-level staff, not just the single username we assigned earlier.

You can then use the command *enable 10*, which has its own password, to reach this new level:

```
Router1>enable 10
Password: <lvl10passwd>
Router1#show ip route
Codes: C - connected, S - static, I - IGRP, R - RIP, M - mobile, B - BGP
       D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
       E1 - OSPF external type 1, E2 - OSPF external type 2, E - EGP
       i - IS-IS, L1 - IS-IS level-1, L2 - IS-IS level-2, ia - IS-IS inter area
       * - candidate default, U - per-user static route, o - ODR
       P - periodic downloaded static route

Gateway of last resort is 172.22.1.3 to network 0.0.0.0

C        172.22.1.0 is directly connected, FastEthernet1/0
O*E1 0.0.0.0/0 [110/3] via 172.22.1.3, 1w2d, FastEthernet1/0
Router1#disable 1
Router1>show ip route
               ^
% Invalid input detected at '^' marker.

Router1>
```

To access the new privilege level, this user used the *enable* command with the optional privilege level keyword 10. The router then prompted her for the level 10 password; after entering it correctly she was allowed to use the *show ip route* command. Finally, she reduced her privilege level back to default user-level (privilege level 1), where the*show ip route* command no longer works.

3.22.4 See Also

Recipe 3.1; Recipe 3.2; Recipe 3.21; Recipe 3.23

Top

Recipe 3.23 Defining Per-Port Privileges

3.23.1 Problem

You want to set the privilege level according to which port you use to access the router.

3.23.2 Solution

To configure the privilege level of a particular line, use the following configuration command:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#line aux 0
Router1(config-line)#privilege level 5
Router1(config-line)#exit
Router1(config)#privilege exec level 5  show ip route
Router1(config)#privilege exec level 1  show ip
Router1(config)#privilege exec level 1  show
Router1(config)#end
Router1#
```

3.23.3 Discussion

By default, every access line has a privilege level of 1. You can change the privilege level assigned to a particular line with the *privilege level* command. The following example shows what happens when we connect to an AUX port that is configured with privilege level 5:

```
Press RETURN to get started.

Router1#show privilege
Current privilege level is 5
Router1#show ip route
Codes: C - connected, S - static, I - IGRP, R - RIP, M - mobile, B - BGP
       D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
       E1 - OSPF external type 1, E2 - OSPF external type 2, E - EGP
       i - IS-IS, L1 - IS-IS level-1, L2 - IS-IS level-2, ia - IS-IS inter area
       * - candidate default, U - per-user static route, o - ODR
       P - periodic downloaded static route

Gateway of last resort is 172.22.1.3 to network 0.0.0.0
```

```
C        172.22.1.0 is directly connected, FastEthernet1/0
O*E1 0.0.0.0/0 [110/3] via 172.22.1.3, 1w2d, FastEthernet1/0
Router1#disable
Router1>show ip route
                  ^
% Invalid input detected at '^' marker.
Router1>
```

Note that no username or password is needed to log in, and the privilege level defaults to 5. This permits us to issue a *show ip route* command. We have raised the privilege of this command to the same level, so it works. When we use the *disable* command to set the privilege level back to 1 and attempt to issue the *show ip route* command again, it fails.

Although we have just shown how to increase the privilege level of a router port, this command is more commonly used to lower the level to 0. Lowering the privilege level provides greater security on insecure lines as well as greater flexibility in restricting commands. For instance, you can use this method to restrict the commands available to a user connected on a particular port down to just *telnet*, preventing all other commands. You can accomplish this by configuring a port to privilege level 0 and lowering the privilege level of the *telnet* command to the same level. This is useful when the router is acting as a terminal server.

3.23.4 See Also

Recipe 3.21; Recipe 3.22

Top

Chapter 4. TACACS+

Top

Introduction

The Terminal Access Controller Access Control System (TACACS) protocol dates back to an earlier era in networking when terminal servers were common. The terminal server was also called a Terminal Access Controller (TAC), so TACACS was the TAC Access Control System.

A company called BBN developed the TACACS protocol in the early 1980s. BBN played a key role in the early development of the Internet (parts of BBN were subsequently absorbed by companies such as Verizon and Cisco). The original protocol included only basic functionality to forward login credentials to a central server, and the ability for the server to respond with a pass or fail based on those credentials.

Cisco implemented several extensions to the original TACACS protocol in 1990, and called the new version XTACACS (Extended TACACS), which is described in RFC 1492. However, the IETF considers this RFC to be purely informational, and not an official protocol specification.

More recently, Cisco has replaced both of these earlier versions of TACACS with a newer implementation called TACACS+. The three different versions are not compatible with one another. In fact, Cisco considers the two earlier versions to be obsolete and no longer supports them, although they are still included in the IOS for backward compatibility reasons. This chapter focuses on only the newest TACACS+ version. There is no RFC protocol specification for TACACS+.

It is important to remember that TACACS+ is a Cisco proprietary standard, unlike the competing Remote Authentication Dial In User Service (RADIUS) protocol, which is an open standard documented in RFC 2865. However, Cisco strongly recommends using TACACS+ instead of RADIUS, and we support this recommendation. Cisco's TACACS+ support is far more mature and robust than RADIUS. Another commonly cited reason for using TACACS instead of RADIUS is the transport model.

TACACS+ uses a TCP transport on port 49, which makes it more reliable than RADIUS, which uses UDP. RFC 2865 includes a lengthy technical defense of the RADIUS UDP implementation. However, TACACS+ and RADIUS use different implementation models. TACACS+ prefers to achieve reliable delivery of data between the client and server, while RADIUS prefers a stateless model that allows it to quickly switch to a backup server.

But there are also more tangible benefits to using TACACS+. The biggest real advantage is that TACACS+ allows true command authorization. This means that you can create very clear usage policies with TACACS+, where different users have access to different commands with very fine administrative granularity. TACACS+ can do this because it separates authentication and authorization functions, while RADIUS can't because it combines them.

Another important advantage is that TACACS+ encrypts the entire payload of the client/server exchange. This is important in highly secure environments. RADIUS, on the other hand, encrypts only the password, so intercepting packets can reveal important information.

The strongest point in favor of RADIUS is the fact that it is an open standard implemented by many vendors, including Cisco. Therefore, if you operate a multivendor network that already includes RADIUS, you may prefer to use RADIUS with your Cisco routers. This chapter does not specifically cover RADIUS, although many of the concepts discussed here are equally applicable to both TACACS+ and RADIUS.

TACACS+ is part of Cisco's AAA framework and works with each of these three functions separately:

Authentication

: Identifies users by challenging them to provide a username and password. This information can be encrypted if required, depending on the underlying protocol.

Authorization

: Provides a method of authorizing commands and services on a per user profile basis.

Accounting

: Accounting functions collect detailed system and command information and store it on a central server where it can be used for security and quality assurance purposes.

Throughout this chapter, we will discuss some of the most important benefits of using centralized AAA services with TACACS+. These include the ability to administer login IDs from a central server, as well as centrally defining login and command authorizations for each user. This allows for easy grouping of users by their administrative functions. For example, you can give network operators access to one set of commands, web site administrators access to a different set, and still allow network engineers to have full access. In addition, you can centrally define and modify these capabilities so that a particular user has similar capabilities on all routers, without having to configure this separately on each router.

Top

Recipe 4.1 Authenticating Login IDs from a Central System

4.1.1 Problem

You want to administer login ID and password information centrally for all routers.

4.1.2 Solution

Cisco changed the AAA syntax slightly in Version 12.0(5)T. The following set of commands allows you to configure TACACS+ authentication in the older (pre-12.0(5)T) IOS versions:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#aaa new-model
Router1(config)#aaa authentication login default tacacs+
Router1(config)#aaa authentication enable default tacacs+
Router1(config)#tacacs-server host 172.25.1.1
Router1(config)#tacacs-server key COOKBOOK
Router1(config)#end
Router1#
```

Newer IOS versions require the *group* keyword, which defines server groups. Therefore, you would now configure the same functionality as follows:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#aaa new-model
Router1(config)#aaa authentication login default group tacacs+
Router1(config)#aaa authentication enable default group tacacs+
Router1(config)#tacacs-server host 172.25.1.1
Router1(config)#tacacs-server key COOKBOOK
Router1(config)#end
Router1#
```

4.1.3 Discussion

When you configure AAA authentication on a router, it starts to ignore the locally configured passwords in favor of those provided by the TACACS+ server. In this example, we have configured the router to consult TACACS+ for both the login and enable passwords. This is a great labor saver because it means that you don't have to reconfigure all of your routers just because you want to change passwords. Instead, because the passwords are stored on a central server, you can change them once

and the new passwords instantly propagate to all of your routers. If the router can't reach the TACACS+ server due to a failure of either the network or the server, it resorts to using the locally configured passwords.

For audit and control reasons, most organizations that implement AAA supply a unique username and password for each individual user. While it is possible to store all of this information locally on the router, if you have a large number of routers, it is extremely time consuming to reconfigure all of the routers to reflect a password change, or to simply add a new user. One of the main advantages to using TACACS+ for AAA authentication is that none of the information is stored on the router. Instead, when a user tries to log in, the router automatically sends a query to the TACACS+ server to verify the login credentials. This minimizes the configuration on each router. And, because this query is done each time, the information is always up-to-date.

When TACACS+ is working correctly, the router prompts for both a login ID and password instead of the usual line password only:

```
freebsd% telnet toronto
Trying 172.25.1.5...
Connected to toronto.
Escape character is '^]'.


User Access Verification

Username: ijbrown
Password: xxxxxxxx

Router1>
```

The most obvious drawback to using a central server for authentication is that it represents a single point of failure. Therefore, TACACS+ allows you to configure several servers:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#tacacs-server host 172.25.1.1 key COOKBOOK
Router1(config)#tacacs-server host 10.12.1.33 key OREILLY
Router1(config)#end
Router1#
```

Note that we have defined different encryption keys for each server. This is the key that TACACS+ uses to encrypt the session between the router and the central server. It is important to protect this encryption key. The ability to configure different keys for the different servers helps to improve your overall security by making sure that you can always switch quickly to the backup server if you suspect that the primary's encryption key has been compromised.

The order of these server commands is important because it reflects the order that the router uses to consult the servers. If the first server is unreachable, the router resorts to the next one, and so on. If no

server responds, the router will use locally configured passwords. This also allows you to easily set up a simple form of load sharing among multiple servers by making one group of routers use the first server as their primary, and having the second group of routers use the second server. Then you can configure both groups of routers to use the other server as a backup. In this way, you can have all of the benefits of fault tolerance as well as load balancing.

The examples in this recipe and many others throughout this chapter show two sets of syntaxes because Cisco changed the AAA commands in IOS Version 12.0(5)T. The big change is the addition of AAA server groups. In the recipe example, we have opted to use the default TACACS+ group, which consists of all of the servers defined using *tacacs-server host* commands:

```
Router1(config)#aaa authentication login default group tacacs+
Router1(config)#aaa authentication enable default group tacacs+
Router1(config)#tacacs-server host 172.25.1.1
```

However, some organizations are so large that they have to deploy many TACACS+ servers. In this case, it is convenient to create groups of servers, either by geography or some other logical grouping:

```
Router1(config)#aaa group server tacacs+ SERVERGROUP-A
Router1(config-sg-tacacs+)#server 172.25.1.1
Router1(config-sg-tacacs+)#server 10.12.1.33
Router1(config-sg-tacacs+)#exit
Router1(config)#aaa authentication login default group SERVERGROUP-A
```

You can also create groups of RADIUS servers if required.

By default, the router allows three login attempts before dropping a session. You can modify this limit using the TACACS+ command *tacacs-server attempts*. In the following example, we have configured the router to allow only one failed login attempt before dropping the session:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#tacacs-server attempts 1
Router1(config)#end
```

Once you implement this command, the router's login behavior will change:

```
freebsd% telnet toronto
Trying  172.25.1.5...
Connected to toronto.
Escape character is '^]'.

User Access Verification

Username: ijbrown
Password: <wrong password>
Connection closed by foreign host.
freebsd%
```

You can configure the maximum number of failed login attempts to be any number between 1 and 1000. However, having a high number makes it considerably easier to launch a brute force password-guessing attack. So in general it is better to keep the maximum number small.

Most large organizations have a security policy that dictates the maximum number of failed logins, with typical values being three or four attempts. Check with your local security department to see what policies you should be following.

4.1.4 See Also

Recipe 4.3; Recipe 4.9; Recipe 3.1

Top

Recipe 4.2 Restricting Command Access

4.2.1 Problem

You want to restrict permissions so that specific users can only use certain commands.

4.2.2 Solution

You can enable TACACS+ command authorization in newer IOS versions with the following set of configuration commands:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#aaa new-model
Router1(config)#aaa authorization exec default group tacacs+
Router1(config)#aaa authorization commands 15 default group tacacs+
Router1(config)#tacacs-server host 172.25.1.1
Router1(config)#tacacs-server key COOKBOOK
Router1(config)#end
Router1#
```

In any IOS version before 12.0(5)T, the AAA syntax was slightly different:

```
Router2#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router2(config)#aaa new-model
Router2(config)#aaa authorization exec default tacacs+
Router2(config)#aaa authorization commands 15 default tacacs+
Router2(config)#end
Router2#
```

4.2.3 Discussion

After you configure AAA authorization, the router queries the TACACS+ server each time somebody enters a command to see if it is allowed. If the user is permitted to use this particular command, the TACACS+ server will respond with an "accept" message and the router will proceed to execute the command. However, if the user is not permitted to issue the command, the TACACS+ server responds with a "reject" message, and the router will not execute the command. The router also shows a rejection status message on the screen:

```
Router1#configure terminal
```

```
Command authorization failed.

Router1#
```

In this case, the current user is unable to modify the router configuration because of an AAA authorization rejection.

Command authorization is useful in many situations. For example, you can use it to allow novice users to access some commands on the router, while preventing them from modifying the configuration. Or, in other cases, you might need to give special access to different groups of users according to their job functions. For example, the night operator might need to have access to look at the routing tables. But you may not want to give this person the same command set that your network engineers have. In Recipe 4.9, we illustrate how to configure a TACACS+ server to permit and deny specific commands.

In the recipe examples, we have configured the router to authorize enable level commands only, by specifying the number 15 as an argument:

```
Router1(config)#aaa authorization commands 15 default tacacs+
```

You may recall from Chapter 3 that the enable level commands are assigned level 15, whereas VTY level commands are at level 1. You can authorize all level 1 commands as well, depending on the level of security and control you wish to enforce. You could authorize all level 1 commands as follows:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#aaa new-model
Router1(config)#aaa authorization commands 1 default tacacs+
Router1(config)#end
Router1#
```

4.2.4 See Also

Recipe 4.3; Recipe 4.9; Chapter 3

Top

Recipe 4.3 Losing Access to the TACACS+ Server

4.3.1 Problem

You want to ensure that your router can still authenticate user sessions, even if it loses access to the TACACS+ server.

4.3.2 Solution

It is important to make sure that you can still enter commands on your router if your TACACS+ server becomes unreachable for any reason. The following set of commands ensures that you don't lose functionality just because you lose your server connection:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#aaa new-model
Router1(config)#aaa authentication login default group tacacs+ enable
Router1(config)#aaa authentication enable default group tacacs+ enable
Router1(config)#aaa authorization commands 15 default group tacacs+ if-authenticated
Router1(config)#tacacs-server host 172.25.1.1
Router1(config)#tacacs-server key COOKBOOK
Router1(config)#end
Router1#
```

4.3.3 Discussion

One of the potential problems with using a central server to authenticate user access is the issue of what happens when you lose access to that server. It would not be terribly useful if you couldn't plug in a console device and reconfigure the router to fix the problem that caused the router to lose access in the first place. But, by default, a router that can't communicate with its AAA server can't authenticate or authorize users.

Fortunately, Cisco's AAA implementation also includes the ability to perform authentication locally on the router in case it can't reach its TACACS+ server. Cisco documentation often refers to this authentication as the "password of last resort." The various authentication methods available within the AAA feature set are shown in Table 4-1.

Table 4-1. AAA authentication methods

Keyword	Definition
tacacs+	TACACS+ authentication
radius	RADIUS authentication
line	Line-based authentication (password)
local	Local username authentication
local-case	Case-sensitive local authentication
enable	Enable password or enable secret
none	No authentication

The example in this recipe shows how to use the router's enable password as a redundant authentication method by adding the keyword *enable* to the *aaa authentication* command. As long as the primary authentication method (TACACS+ in this case) is working, the router never uses this password of last resort. However, when the server connection is lost, users will be prompted for the enable password instead of the TACACS+ username and password. This ensures that you will never be locked out of your routers.

You can also implement other backup authentication methods such as local authentication, line-based passwords, and even RADIUS. However, we recommend using the combination of the enable password method shown in this recipe along with using an enable secret password for two reasons. First, this password is local to the router so it will never become unavailable. Second, when you use enable secret passwords, the router stores the password using MD5 encryption internally, which will help protect it from prying eyes. We should also mention that it is possible to string together a few different methods of authentication, although this is usually unnecessary.

This example assumes that we are doing command authorization as well as authentication. The same problems that we just mentioned for authentication also apply to authorization. It doesn't do you any good to get into the router if the router can't verify which commands you are authorized to use. This is why we have included the *if-authenticated* keyword in the aaa authorization command:

```
Router1(config)#aaa authorization commands 15 default group tacacs+ if-authenticated
```

We highly recommend using the *if-authenticated* option whenever you enable AAA authorization.

4.3.4 See Also

Recipe 4.1 ; Recipe 4.2

Top

Recipe 4.4 Disabling TACACS+ Authentication on a Particular Line

4.4.1 Problem

You want to disable TACACS+ authentication on your router's console interface.

4.4.2 Solution

You can disable TACACS+ authentication on the router's console port while leaving it active on the rest of the router lines:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#aaa new-model
Router1(config)#aaa authentication login default group tacacs+ local
Router1(config)#aaa authentication login OREILLY line
Router1(config)#line con 0
Router1(config-line)#login authentication OREILLY
Router1(config-line)#end
Router1#
```

4.4.3 Discussion

By default, when you configure a router to use AAA authentication, it automatically applies this authentication method to all lines. This means that you don't have to explicitly configure each line to use AAA authentication. Normally this default behavior is useful because it requires less configuration. But there are times when you may want to use different authentication methods on different lines. For instance, in our example we wanted to be able to access the router's console line with a simple password. But we didn't want this change to affect the AAA authentication on any of the VTY or AUX lines.

The first two lines in the example simply enable TACACS+ authentication for all login access to the router:

```
Router1(config)#aaa new-model
Router1(config)#aaa authentication login default group tacacs+ local
```

As soon as you enter these commands, every line on the router, including the console, will begin to use TACACS+ for authentication. The next command creates a new AAA authentication group called *OREILLY* that uses the local line password for authentication:

```
Router1(config)#aaa authentication login OREILLY line
```

This command doesn't do anything yet, though, because none of the router's lines belongs to this new authorization group. We have to configure the console line with the *login authentication* command to associate this line with the authentication group:

```
Router1(config)#line con 0
Router1(config-line)#login authentication OREILLY
```

Now, when a user connects on the console, they will use the type of authentication specified for this group. In this case, if you look back at the group definition, you will see that the OREILLY group uses *line* authentication. However, because we have associated only the console line with this group, all of the other lines will continue to use the TACACS+ authentication method.

If you wanted to, you could configure a different group for every line. But, in general, we recommend using the default TACACS+ authentication method on all lines, even the console, unless there is a compelling reason to do otherwise. You don't need to worry about losing console access because of a problem on the central server; you can always implement a password of last resort, as described in Recipe 4.3.

You can return the console to the default authentication group by simply changing the login authentication line again:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#line con 0
Router1(config-line)#login authentication default
Router1(config-line)#end
Router1#
```

4.4.4 See Also

Recipe 4.1; Recipe 4.3

Top

Recipe 4.5 Capturing User Keystrokes

4.5.1 Problem

You want to capture and timestamp all keystrokes typed into a router and associate them with a particular user.

4.5.2 Solution

The AAA accounting feature allows you to capture keystrokes and log them on the TACACS+ server:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#aaa new-model
Router1(config)#aaa accounting commands 1 default stop-only group tacacs+
Router1(config)#aaa accounting commands 15 default stop-only group tacacs+
Router1(config)#end
Router1#
```

4.5.3 Discussion

The ability to capture every keystroke entered into a router is a powerful security and quality assurance feature that is extremely useful. For instance, keystroke logging is extremely useful in forensic reconstruction of network events. TACACS+ provides the ability to capture all keystrokes typed into your routers and log them for future reference. The TACACS+ log contains the command that was typed, along with other useful information such as time and date, router name, username, originating IP address, and privilege level. Here is an example of a TACACS+ accounting record:

```
Fri Jan  3 11:08:47 2003        toronto ijbrown  tty66   172.25.1.1
stop    task_id=512 start_time=1041610127   timezone=EST    service=shell
priv-lvl=15      cmd=configure terminal <cr>
```

In this log entry we can see that user *ijbrown* submitted the command *configure terminal* on router *toronto* at 11:08 on January 3, 2003. It also shows that this user accessed the router from IP address `172.25.1.1` using *tty66*.

To save disk space on your TACACS+ server, you may decide to log only level 15-based commands, which is done with this command:

```
Router1(config)#aaa accounting commands 15 default stop-only group tacacs+
```

Level 1 commands are generally relatively benign and pose little real threat to the security or health of the router. So logging them is less important than for level 15 commands. But we generally recommend logging all commands if you're logging commands at all because the level 1 commands might show useful patterns of information. You can also log the commands issued at any other user level by adding more *aaa accounting* lines and specifying the appropriate user level.

4.5.4 See Also

Recipe 4.6; Recipe 4.9

Top

Recipe 4.6 Logging System Events

4.6.1 Problem

You want to log various system events.

4.6.2 Solution

AAA accounting includes the ability to log a variety of system events, including timestamps along with associated usernames:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#aaa new-model
Router1(config)#aaa accounting exec default start-stop group tacacs+
Router1(config)#aaa accounting connection default start-stop group tacacs+
Router1(config)#aaa accounting system default stop-only group tacacs+
Router1(config)#end
Router1#
```

4.6.3 Discussion

In addition to capturing keystroke logs, AAA accounting can gather other useful pieces of information such as EXEC, connection, and system events.

exec

> This feature captures and timestamps the beginning and ending of a user's EXEC session on the router.

connection

> This feature allows you to gather information about outgoing connections using an interactive protocol such as Telnet, SSH, or RSH.

system

> When you enable this feature, AAA forwards information about system events such as router reboots or the disabling of AAA accounting.

Here is an example of an EXEC log entry:

```
Fri Jan  3 11:11:40 2003         toronto ijbrown tty67   172.25.1.1
start    task_id=514 start_time=1041610300    timezone=EST     service=shell
```

```
Fri Jan  3 11:18:47 2003         toronto ijbrown tty67   172.25.1.1
stop     task_id=514 start_time=1041610300   timezone=EST     service=shell
disc-cause=1    disc-cause-ext=1020 connect-progress=101     elapsed_time=427
nas-rx-speed=0   nas-tx-speed=0
```

These two records show that user *ijbrown* logged into router *toronto* at 11:11:40 AM on January 3, 2003, and stayed connected for 427 seconds. This information is useful for security auditing, and also can be used for billing purposes if required. For example, if you are using this router to provide PAD or terminal server services to paying customers, this is an ideal way to gather billing information.

Here is an example of a connection log event:

```
Fri Jan  3 11:30:19 2003         toronto ijbrown tty67   172.25.1.1
stop     task_id=522start_time=1041611404    timezone=EST    service=connection
protocol=telnet addr=10.2.2.2 cmd=telnet 10.2.2.2     pre-bytes-in=0  pre-bytes-out=0
pre-paks-in=0    pre-paks-out=0  bytes_in=1843  bytes_out=81  paks_in=43   paks_out=50
connect-progress=47    elapsed_time=15 nas-rx-speed=0   nas-tx-speed=0
```

In this record you can see that user *ijbrown* initiated a Telnet session to IP address `10.2.2.2`, and terminated it 15 seconds later. You can even see the total number of bytes and packets both sent and received by the Telnet process.

The system event log entries look like this:

```
Fri Jan  3 11:35:19 2003         toronto unknown unknown unknown stop    task_id=265
start_time=1041611719    timezone=EST    service=system  event=sys_acct
reason=shutdown

Fri Jan  3 11:37:35 2003         toronto unknown unknown unknown start    task_id=1
timezone=EST    service=system  event=sys_acct  reason=reload
```

These records show that somebody reloaded the router called *toronto* at 11:35 on January 3, 2003. It came back up at 11:37 some two minutes later. Notice that the system event logging did not capture information on the user who submitted the *reload* command. That information could be captured using command logging.

Actually, this points out an interesting side benefit to capturing this information on a central server. If you were just using regular system logging in the router's log buffer, this information would be lost during the reboot. However, by storing system events on the TACACS+ server, you don't lose anything when the router reboots.

4.6.4 See Also

Recipe 4.5 ; Recipe 4.9

Top

Recipe 4.7 Setting the IP Source Address for TACACS+ Messages

4.7.1 Problem

You want the router to use a particular source IP address when sending TACACS+ logging messages.

4.7.2 Solution

The *ip tacacs source-interface* configuration command allows you to specify a particular source IP address for TACACS logging messages:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#ip tacacs source-interface loopback0
Router1(config)#end
Router1#
```

Note that implementing this command will not only affect AAA accounting, it will also affect AAA authentication and AAA authorization.

4.7.3 Discussion

Normally, when you enable TACACS+ on a router, the source IP addresses on the messages that it sends to the TACACS+ server will be the address of the router's nearest interface. However, this is not always meaningful. If there are many different paths to the server, the router could wind up sending messages through different interfaces. On the server, then, these messages will usually look like they came from different routers, which can make it difficult to analyze the logs.

However, if you use a *loopback* address for the source, all messages from this router will look the same, regardless of which interface they were delivered through. In many networks, the DNS database contains only these loopback IP addresses, which helps to make the logs more useful as well.

We strongly recommend using this command.

4.7.4 See Also

Recipe 4.5; Recipe 4.6

Top

Recipe 4.8 Obtaining Free TACACS+ Server Software

4.8.1 Problem

You are looking for TACACS+ server software for use in your network.

4.8.2 Solution

Cisco distributes a free TACACS+ software system from their anonymous FTP site on the Internet:

```
freebsd% ftp ftp-eng.cisco.com
Connected to ftp-eng.cisco.com.
220 ftp-eng.cisco.com FTP server (Version wu-2.6.2(1) Thu Dec 13 23:14:01 PST 2001)
ready.
Name (ftp-eng.cisco.com:ijbrown): ftp
331 Guest login ok, send your complete e-mail address as password.
Password: <email@address>
230 Guest login ok, access restrictions apply.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> cd pub/tacacs
250-Please read the file README
250-  it was last modified on Sun Jun 18 10:29:35 2000 - 927 days ago
250 CWD command successful.
ftp> ls
200 PORT command successful.
150 Opening ASCII mode data connection for /bin/ls.
total 544
drwxr-x--x   2 1500     eng          4096 Dec  4  1998 RCS
-rw-r--r--   1 1500     eng          1602 Jun 18  2000 README
-r--r--r--   1 1500     eng         73880 Dec  4  1998 tac-rfc.1.78.txt
-rw-rw-r--   1 1500     eng        193771 Jun 18  2000 tac_plus.F4.0.4.alpha.tar.Z
226 Transfer complete.
ftp> bin
200 Type set to I.
ftp> get tac_plus.F4.0.4.alpha.tar.Z
local: tac_plus.F4.0.4.alpha.tar.Z remote: tac_plus.F4.0.4.alpha.tar.Z
200 PORT command successful.
150 Opening BINARY mode data connection for tac_plus.F4.0.4.alpha.tar.Z (193771
bytes).
226 Transfer complete.
193771 bytes received in 1.81 seconds (104.58 KB/s)
ftp> quit
221-You have transferred 193771 bytes in 1 files.
221-Total traffic for this session was 196024 bytes in 2 transfers.
221-Thank you for using the FTP service on ftp-eng.cisco.com.
```

```
221 Goodbye.
freebsd%
```

4.8.3 Discussion

Cisco offers a free version of TACACS+ that you can download. However, please note that it is not a fully supported version. In fact, Cisco seldom updates the free TACACS+ daemon and it lacks several of the advanced features of commercially available TACACS+ software. Furthermore, Cisco warns that this free software comes with no warranty or support.

With this in mind however, the free TACACS+ software is useful for testing, lab environments, or even small installations. You will find that the free Unix version is fully functional and provides an excellent range of features. In fact, all of the AAA accounting log files in this chapter were captured using Cisco's free TACACS+ software.

Please follow the instructions that Cisco provides with the software to extract, compile, and install the TACACS+ server software. Recipe 4.9 includes some sample configuration files.

4.8.4 See Also

Recipe 4.9

Top

Recipe 4.9 Sample Server Configuration Files

4.9.1 Problem

You want to configure a TACACS+ server to accept AAA requests from your network devices.

4.9.2 Solution

Here is an example of a TACACS+ server configuration file that accepts AAA requests from network devices to authenticate users. You can use this example as a template to help you build your own configuration files:

```
freebsd% cat tac.conf
key = "COOKBOOK"

accounting file = /var/log/tacacs

user = ijbrown {
    default service = permit
    member = staff
    login = cleartext cisco
}

user = kdooley {
    default service = permit
    member = staff
     login = des l5c2fHiF21uZ6
}

user = $enab15$ {
    login = cleartext happy
}


group = staff {
    # Default Group
}
```

4.9.3 Discussion

In this recipe, we look at how to configure Cisco's free TACACS+ server software because we want to show how the TACACS+ server works. Most of the configuration is done at the central server, so

understanding a basic configuration helps with understanding AAA services in general. Please note that other TACACS+ servers use different configuration syntax; however, the basic concepts are the same.

The first thing you need to configure is the TACACS+ encryption key. This key must be identical to the one configured on your router configuration with the *tacacs-server key* command. If the keys are not identical, none of the TACACS+ services will work. In the following example, we use the same encryption key as in Recipe 4.1, *COOKBOOK*:

```
key = "COOKBOOK"
```

To configure authentication for a single user, define a username and password combination as follows:

```
user = ijbrown {
        login = cleartext cisco
}
```

In this example, we configured the username *ijbrown* and assigned it a password, *cisco*. If you prefer, you can encrypt the password using DES encryption and store only this encrypted form in the configuration file. However, for this example, we chose to use a clear-text password. The TACACS+ server is now ready to accept authentication requests for this user.

If you choose to use TACACS+ to authenticate your enable password as well, you need to define a special enable user called *$enabl15$*. The following example creates this enable account using the password *happy*. After you define this username, the TACACS+ server will be able to handle authentication requests for the enable password:

```
user = $enab15$ {
    login = cleartext happy
}
```

To enable AAA authorization, you need to define command authorization using a series of *grep*-style regular expressions. For more information on regular expressions, see the *egrep* manpage, or Mastering Regular Expressions (O'Reilly).

By default, TACACS+ implicitly denies all commands. So, unless you explicitly allow a user to issue a command, it will be denied. Also note that the router will fully expand all commands before sending them to the TACACS+ server. This means that if a user types in *sh ip rou*, as a short form for *show ip route*, the TACACS+ server is asked to authorize the long form version of the command, *show ip route*. This is important to remember when you build your regular expression command statements.

The following example shows how to enable command authorization using the *cmd* statement. This configuration example permits the user *sarnott* to issue all *show* commands with the exception of *show running-config*:

```
user = sarnott {
    member = staff
    login = cleartext nssor
```

```
    cmd = show {
        deny running-config
        permit .*
    }
}
```

Notice that we have included two instructions in the *cmd = show* section. The first command, *deny running-config*, denies the user access to the *show running-config* command, while the permit .* line enables access to all other *show* commands. This works because the regular expression `.*` matches everything.

We mentioned earlier that, by default, the TACACS+ server implicitly denies all commands. This is a failsafe approach, but it does make things more difficult when you want to allow a large number of commands. Fortunately, you can use the *default service = permit* command to change this default behavior so that TACACS+ will permit any commands that you don't explicitly deny. When you include this command in a user's configuration, they can issue any command they like. If there are certain commands that you don't want them to use, you can use a *deny* statement to prevent the specific commands, as follows:

```
user = ijbrown {
    default service = permit
    login = cleartext cisco
    cmd = debug {
        deny .*
    }
}
```

Here we have configured user *ijbrown* so that, by default, he is permitted to access all commands. We then specifically denied access to the *debug* commands using the *cmd* statement. You can include several such *cmd* statements, if required. This illustrates one of TACACS+'s greatest strengths: the fine granularity of its command authorization.

TACACS+ also allows you to put users into TACACS+ groups for easier administration. You can define several different profile groups and assign users to them as required. For example, you might create a group for users who are only allowed to look at show commands, another group for users who can do everything except debugging, and an administration group with full access to everything. You simply define a group, build its attributes, and assign users to the group. These users will inherit the group attributes, which will keep administration to the minimum.

Our next example creates a group and assigns a user to the new group:

```
user = kdooley {
    default service = permit
    member = staff
     login = des l5c2fHiF21uZ6
}
```

```
group = staff {
    # Default Group
    cmd = debug {
        deny .*
    }
}
```

In this example, we have assigned the user *kdooley* to the group called *staff*, using the *member* command. This new user will now inherit the group's attributes. In this example, all other members of the *staff* group, including user *kdooley*, are allowed to use any command except the *debug* commands.

Finally, if you wish to enable AAA accounting, you need to define the log file in which TACACS+ will store these accounting messages:

```
accounting file = /var/log/tacacs
```

Interestingly enough, this is the only configuration command required to enable accounting on the server. As soon as you define this accounting file, the TACACS+ server is able to begin capturing these messages from any routers that are configured to send them.

AAA accounting logs can grow rather unruly, especially if you are using command logging. So we highly recommend cycling your accounting log on a daily basis, keeping at least a week's worth of logs in case of emergencies. Recipe 18.11 shows a script that will rotate your accounting log files.

4.9.4 See Also

Recipe 4.1; Recipe 4.2; Recipe 4.5; Recipe 18.11; Mastering Regular Expressions, Second Edition, by Jeffrey E.F. Friedl (O'Reilly)

Top

Chapter 5. IP Routing

Top

Introduction

IP routing works by comparing the destination addresses of IP packets to a list of possible destinations called the routing table. The destination address in a packet usually identifies a single host. It is also possible to use the multicast functions of the IP protocol to send packets to many hosts simultaneously, as discussed in Chapter 23. In this chapter, however, we focus on routing to one specific destination, which is called *unicast* routing.

In a very large network, such as the public Internet or a large corporate network, it is impractical to keep track of every individual device. Instead, the IP protocol groups devices together into subnets. A subnet is, in effect, a summary address representing a group of adjacent hosts. And, similarly, you can summarize adjacent groups of subnet addresses. The result is an extremely efficient hierarchical addressing system.

There are two different sets of rules for how groups of subnets can be summarized together. The older method uses a concept called *class*, while the newer method is *classless* and is often referred to by the acronym CIDR, for Classless Inter-Domain Routing. CIDR is described in detail in RFCs 1517, 1518 and 1519. Both methods are still in common use, although the public Internet makes extensive use of CIDR, and all newly registered IP addressing follows the new rules.

You can turn on CIDR in Cisco routers with the global configuration command *ip classless*. Classless routing has been the default since IOS Version 11.3. If the older rules are required, you have to explicitly disable CIDR with the *no ip classless* command.

For small networks, the distinction is often irrelevant, particularly if they don't use a dynamic routing protocol. However, using a mixture of classful and classless addressing and routing models in a network can cause some extremely strange and unexpected routing behavior. Because many network administrators are unclear on the distinctions, a brief review is in order.

The biggest difference between classful and classless addressing is that classful addressing assumes that the first few bits of the address can tell you how big the network is. Table 5-1 shows how address classes are defined. As you can see, a Class A address is any network from `0.0.0.0` to `127.0.0.0`, and all of these networks are assumed to have a mask of `255.0.0.0` (`/8`).

Table 5-1. Classes of IP addresses

Class	Range of network addresses	Mask	Mask bits
A	0.0.0.0 - 127.0.0.0	255.0.0.0	8
B	128.0.0.0 - 191.255.0.0	255.255.0.0	16
C	192.0.0.0 - 223.255.255.0	255.255.255.0	24
D	224.0.0.1 - 239.255.255.255	255.255.255.255	32
E	240.0.0.1 - 255.255.255.255	255.255.255.255	32

You can create several subnets within a Class A, B, or C network. However, it is harder to work with structures that are larger than the network. For example, if you wanted to work with the networks `192.168.4.0/24`, `192.168.5.0/24`, `192.168.6.0/24`, and `192.168.7.0/24`, CIDR would allow you to address this entire group (called a *supernet*) as `192.168.4/22` (or `192.168.4.0  255.255.252.0` in netmask notation). However, with classful routing, the router would have to maintain routes to all of these ranges as separate Class C networks.

A router decides where to send a packet by comparing the destination address in the header of the IP packet with its routing table. The rule is that the router must always use the most specific match in the table. This will be the entry that has the most bits in its netmask, so it is often called the *longest* match. This longest match rule is required because the routing table often contains several possible matches for a particular destination.

For example, suppose the destination address in a particular packet is `10.5.15.35`. The router will look in its routing table for possible matches and the accompanying next-hop information that will tell it where to send this packet. If there is a match for the specific host, `10.5.15.35/32`, it doesn't need to look any further. But, it is more likely that the router will find a more general route, such as `10.5.15.0/24` or `10.5.0.0/16`. And, if it can't find any reasonable matches, there is usually a *default route* or *gateway of last resort*, `0.0.0.0/0`, that matches anything. If there is no match at all, the router must drop the packet.

Classless routing can use a mask of any length when looking for the best route to a destination, but classful routing cannot. For example, CIDR would allow the four networks `192.168.4.0/24`, `192.168.5.0/24`, `192.168.6.0/24` and `192.168.7.0/24` to be written together as `192.168.4.0/22`. But a router using classful routing would not consider the destination address `192.168.5.15` to be a part of `192.168.4.0/22` because it knows that anything beginning with `192` must be a Class C network. Instead, if there was no specific route for `192.168.5.0/24` or a subnet containing this destination, the router would skip straight to the default route. If you mix classless and classful routing, this could be the wrong path, and in the worst case, it could even cause a routing loop.

This is why it is so important to make sure that you are consistent about which type of routing and addressing you want to use. In general, it is better to use CIDR because of the improved flexibility it offers. Also, since CIDR allows more levels of route summarization, you can often simplify your

routing tables so that they take up less memory in the routers. This, in turn, can improve network performance.

Summary routes have another important benefit. The router will keep its summary route as long as any of its subnets exist. This means that the summary route is as stable as the most stable route in the summarized range. Without summarization, if there is one route that repeatedly flaps up and down, the routing protocol must propagate every transition throughout the network. But a summary route can hide this instability from the rest of the network. The routing protocol doesn't need to waste resources installing and removing the flapping route, which improves overall network stability.

Unregistered Addresses

Most of the IP addresses used in examples in this book are unregistered. The Internet Engineering Task Force (IETF) and the Internet Assigned Numbers Authority (IANA) have set aside several unregistered ranges of addresses for anybody to use at any time. The only stipulation is that, because anybody and everybody is using these numbers, they cannot be allowed to leak onto any public sections of the Internet. The allowed ranges of unregistered IP addresses are defined in RFC 1918 and summarized in Table 5-2. It is a good practice to address all private networks using these address ranges.

Table 5-2. RFC 1918 allowed unregistered IP addresses

Class	Network	Mask	Comment
Class A	10.0.0.0	255.0.0.0	One large Class A network
Class B	172.16.0.0 - 172.31.0.0	255.255.0.0	16 Class B networks
Class C	192.168.0.0 - 192.168.255.0	255.255.255.0	256 Class C networks

Note that RFC 3330 defines a number of other special ranges including a special TEST-NET range, `192.0.2.0/24`, which is reserved for documentation purposes. We occasionally use this address range in this book. You should not use it in production networks, however.

Top

Recipe 5.1 Finding an IP Route

5.1.1 Problem

You want to find a particular route in your router's routing tables.

5.1.2 Solution

The EXEC-level command to look at the entire IP routing table is:

```
Router>show ip route
Codes: C - connected, S - static, I - IGRP, R - RIP, M - mobile, B - BGP
       D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
       E1 - OSPF external type 1, E2 - OSPF external type 2, E - EGP
       i - IS-IS, L1 - IS-IS level-1, L2 - IS-IS level-2, * - candidate default
       U - per-user static route, o - ODR

Gateway of last resort is 172.25.1.1 to network 0.0.0.0

     192.168.17.0/27 is subnetted, 1 subnets
C       192.168.17.0 is directly connected, Loopback1
     172.16.0.0/30 is subnetted, 1 subnets
C       172.16.1.0 is directly connected, Async1
     172.25.0.0/16 is variably subnetted, 6 subnets, 3 masks
C       172.25.25.0/30 is directly connected, Tunnel0
C       172.25.1.0/24 is directly connected, Ethernet0
C       172.25.9.0/24 is directly connected, Ethernet1
C       172.25.10.1/32 is directly connected, Loopback0
O       172.25.100.1/32 [110/11] via 172.25.9.2, 4d09h, Ethernet1
O IA    172.25.100.0/24 [110/11] via 172.25.1.1, 2d11h, Ethernet0
     192.168.1.0/32 is subnetted, 1 subnets
S       192.168.1.1 [1/0] via 172.25.1.4
O*E1 0.0.0.0/0 [110/11] via 172.25.1.1, 1d07h, Ethernet0
```

You can also find the route to a particular device, such as `172.25.100.15`:

```
Router>show ip route 172.25.100.15
Routing entry for 172.25.100.0/24
  Known via "ospf 55", distance 110, metric 11, type inter area
  Redistributing via ospf 55
  Last update from 172.25.1.1 on Ethernet0, 2d12h ago
  Routing Descriptor Blocks:
```

```
  * 172.25.1.1, from 172.25.1.1, 2d12h ago, via Ethernet0
      Route metric is 11, traffic share count is 1
```

5.1.3 Discussion

The output of the first command contains a lot of useful information. At the top is an explanation of the different codes used in the information that follows. For example, every line that begins with a "C" refers to a route that is directly connected to one of the router's interfaces, "S" means a static route, and so forth. Note that it is possible to have more than one such code, as in the route for `172.25.100.0/24`. In this case, the first letter (O) indicates that this route was learned via the OSPF routing protocol. The next two letters (IA) indicate that this is an OSPF inter-area route. Similarly, the entry for `0.0.0.0/0` is an OSPF external route of type 1. The asterisk (*) indicates that this route is a candidate for default route. There may be several such candidates, but the one that the router thinks is best is captured at the top of the table in the line beginning "Gateway of last resort."

In a large network, it is often not very useful to list the entire routing table like this. Instead, if you are looking for a particular route, you can search for it directly, as the second example in this recipe shows.

The router didn't have a 32-bit match for this particular destination (`172.25.100.15`), so it shows its best match, `172.25.100.0/24`. The result shows several useful pieces of information. It says exactly which match is being used. It tells you how the router knows about this route, in this case via OSPF Process ID number 55. It also indicates that the next hop required to reach that destination is the device `172.25.1.1`, and that it reaches this device through the interface `Ethernet0`.

If the desired route cannot be resolved except by means of the default gateway, the response is much shorter:

```
Router> show ip route 172.16.101.5
% Network not in table
```

This means that the router will use the default route when trying to reach this device. If there is no default route, it will drop the packets. Note that this assumes we are using classless routing. If we had enabled classful routing on the router, a route for any subnet of the classful network would also have an entry for the entire network. In this case, the classful network would be `172.16.0.0/16`.

Top

Recipe 5.2 Finding Types of IP Routes

5.2.1 Problem

You want to look for a particular type of route in your router's routing tables.

5.2.2 Solution

Often you are more interested in finding all of the directly connected networks, or all of the static routes, rather than a specific route. This can be done easily by specifying the type of route in the *show* command:

```
Router>show ip route connected
     192.168.17.0/27 is subnetted, 1 subnets
C       192.168.17.0 is directly connected, Loopback1
     172.16.0.0/30 is subnetted, 1 subnets
C       172.16.1.0 is directly connected, Async1
     172.25.0.0/16 is variably subnetted, 6 subnets, 3 masks
C       172.25.25.0/30 is directly connected, Tunnel0
C       172.25.1.0/24 is directly connected, Ethernet0
C       172.25.9.0/24 is directly connected, Ethernet1
C       172.25.10.1/32 is directly connected, Loopback0

Router>show ip route static
     192.168.1.0/32 is subnetted, 1 subnets
S       192.168.1.1 [1/0] via 172.25.1.4
```

Another useful variant of the *show ip route* command summarizes all of the different types of routes in the table:

```
Router>show ip route summary
IP routing table name is Default-IP-Routing-Table(0)
Route Source    Networks    Subnets     Overhead    Memory (bytes)
connected       0           3           328         432
static          1           0           64          144
ospf 55         1           3           256         576
  Intra-area: 1 Inter-area: 2 External-1: 1 External-2: 0
  NSSA External-1: 0 NSSA External-2: 0
internal        2                                   2328
Total           4           6           648         3480
```

5.2.3 Discussion

You can see the full list of possibilities by using a question mark (?) on the command line:

```
Router>show ip route ?
Hostname or A.B.C.D  Network to display information about or hostname
  bgp                  Border Gateway Protocol (BGP)
  connected            Connected
  egp                  Exterior Gateway Protocol (EGP)
  eigrp                Enhanced Interior Gateway Routing Protocol (EIGRP)
  igrp                 Interior Gateway Routing Protocol (IGRP)
  isis                 ISO IS-IS
  list                 IP Access list
  odr                  On Demand stub Routes
  ospf                 Open Shortest Path First (OSPF)
  profile              IP routing table profile
  rip                  Routing Information Protocol (RIP)
  static               Static routes
  summary              Summary of all routes
  supernets-only       Show supernet entries only
  traffic-engineering  Traffic engineered routes
  <cr>
```

This is useful when you want to see all of the routes that the router has learned via a particular routing protocol, or all of the statically configured or directly connected routes. The output format with the different type keywords is exactly the same as for the general *show ip route*, but it presents only the routes of the required type.

The *show ip route summary* command gives useful information about the size of the routing table and how much memory the router has allocated to storing this information, conveniently broken down by routing protocol. The example also shows how many routes belong to each of the different OSPF area types.

This command has several uses. First, it gives you a convenient way to estimate your routing table's memory requirements. In this case, the routing table is very small, so more memory is used to store connected routes than OSPF routes. However, in a larger network you will often want to know if one routing protocol is causing memory problems for your routers. This can help you decide if you need route filtering or summarization mechanisms. Routers exchanging BGP routing information with the public Internet can have particularly serious memory utilization problems.

Second, because it shows how many routes are learned by each mechanism, you can easily check the stability of the routing table by seeing whether this number changes in time. If you look at the entire routing table you may not notice that a handful of routes periodically disappear and reappear, but looking at this summary information makes it much easier to spot such problems.

Third, you can easily see whether your routing table is getting its information the way you expect. It can be a very quick and easy way to check if the router is installing floating static routes or external routes in its routing table.

Top

Recipe 5.3 Converting Different Mask Formats

5.3.1 Problem

You want to convert between the three different formats that Cisco routers use to present mask information: standard netmask, ACL wildcards, and CIDR bit numbers.

5.3.2 Solution

The following Perl script converts from any of these formats to any other. The usage syntax is "mask-cvt {n|w|b} {n|w|b} {nnn.nnn.nnn.nnn|/bits}", where the first argument specifies what the input format is and the second argument specifies the output format. In both cases n is for netmask format, w is for wildcard format, and b is for CIDR bit format (with or without the leading slash, as in `/24`).

For example:

```
$ mask-cvt.pl n w 255.255.248.0
0.0.7.255
$ mask-cvt.pl n b 255.255.248.0
/21
$ mask-cvt.pl w n 0.3.255.255
255.252.0.0
$ mask-cvt.pl w b 0.3.255.255
/14
$ mask-cvt.pl b n /21
255.255.248.0
$ mask-cvt.pl b w /21
0.0.7.255
```

Example 5-1 contains the Perl code for the *mask-cvt* script.

Example 5-1. mask-cvt.pl

```
#!bin/perl
#
#   mask-cvt.pl -- a script to convert between the various
#                  methods of masking IP addresses
#
sub usage(  ) {
   print "mask-cvt [nwb] [nwb] {nnn.nnn.nnn.nnn|bbb}\n";
   print "     where the first argument, [nwba], specifies the input \n";
   print "           format as one of netmask, wildcard or number of \n";
```

```
   print "           bits and the second argument, [nwb], specifies \n";
   print "           the output format\n";

   exit(  );
}

if($#ARGV != 2) { usage(  ); }

# get the input format style
$_ = @ARGV[0];

if(/[nN]/) {
      # incoming format netmask, what's the outgoing
      $_ = @ARGV[1];
      if(/[nN]/) {
         # no conversion
         $output = @ARGV[2];
         } elsif (/[wW]/) {
            # out is wildcard
            $output = do_subtract(@ARGV[2]);
         } elsif (/[bB]/) {
            # out is wildcard
            $output = do_bits(@ARGV[2]);
         } else {
            usage(  );
         }
} elsif (/[wW]/) {
    # incoming format wildcard, what's the outgoing
    $_ = @ARGV[1];
    if(/[wW]/) {
       # no conversion
       $output = @ARGV[2];
       } elsif (/[nN]/) {
          # out is wildcard
          $output = do_subtract(@ARGV[2]);
       } elsif (/[bB]/) {
          # out is wildcard
          $output = do_bits(do_subtract(@ARGV[2]));
       } else {
          usage(  );
       }
} elsif (/[bB]/) {
    # remove any leading "/" in the bit count
    $_ = @ARGV[2];
    s/[-\/]//;
    $bits = $_;

    # incoming format is bit count, what's the outgoing
    $_ = @ARGV[1];
    if(/[bB]/) {
       # no conversion
       $output = @ARGV[2];
```

```
            print "no conversion\n";
            } elsif (/[nN]/) {
                # out is netmask
                $output = cvt_bits_mask($bits);
            } elsif (/[wW]/) {
               # out is wildcard
               $output = do_subtract(cvt_bits_mask($bits));
            } else {
              usage(  );
            }
      } else {
         usage(  );
    }

    print "$output\n";

    sub do_subtract(  ) {
      local($ip) = @_;

      # break up the bytes of the incoming IP address
      $_ = $ip;
      ($a, $b, $c, $d) = split(/\./);

      if ($a > 255 || $b > 255 || $c > 255 || $d > 255 || /[^0-9.]/) {
         print "invalid input mask or wildcard\n";
         exit(  );
      }

      $a = 255 - $a;
      $b = 255 - $b;
      $c = 255 - $c;
      $d = 255 - $d;

      return ($a . "." . $b . "." . $c . "." . $d);
    }

    sub do_bits(  ) {
      local($ip) = @_;

      # break up the bytes of the incoming IP address
      $_ = $ip;
      @ip_bytes = split(/\./);

      if ($ip_bytes[0] > 255 || $ip_bytes[1] > 255 || $ip_bytes[2] > 255
           || $ip_bytes[3] > 255 || /[^0-9.]/ || $#ip_bytes != 3) {
         print "invalid input mask or wildcard\n";
         exit(  );
      }

      $bits = 0;
      for ($i=0; $i < 4 ; $i++) {
         if ($ip_bytes[$i] > 0 && $bits < 8*$i) {
```

```
            print "invalid mask for bit count format\n";
            exit(  );
         }
         if ($ip_bytes[$i] =  = 255 ) { $bits += 8;
         } elsif ($ip_bytes[$i] =  = 254 ) { $bits += 7;
         } elsif ($ip_bytes[$i] =  = 252 ) { $bits += 6;
         } elsif ($ip_bytes[$i] =  = 248 ) { $bits += 5;
         } elsif ($ip_bytes[$i] =  = 240 ) { $bits += 4;
         } elsif ($ip_bytes[$i] =  = 224 ) { $bits += 3;
         } elsif ($ip_bytes[$i] =  = 192 ) { $bits += 2;
         } elsif ($ip_bytes[$i] =  = 128 ) { $bits += 1;
         } elsif ($ip_bytes[$i] != 0 ) {
            print "invalid mask for bit count format\n";
            exit(  );
         }
      }
      return("/" . $bits);
   }
   sub cvt_bits_mask(  ) {
      local($bits) = @_;

      if ($bits <= 8 ) {
         $a = bits_to_dec($bits);
         $b=$c=$d=0;
      } else {
         $a=255;
         if ($bits <= 16 ) {
            $b = bits_to_dec($bits-8);
            $c=$d=0;
         } else {
            $b=255;
            if ($bits <= 24 ) {
               $c = bits_to_dec($bits-16);
               $d=0;
            } else {
               $c=255;
               if ($bits <= 32 ) {
                  $d = bits_to_dec($bits-24);
               } else {
                  print "invalid bit count\n";
                  exit(  );
               }
            }
         }
      }
      return ($a . "." . $b . "." . $c . "." . $d);
   }

   sub bits_to_dec(  ) {
      local($bits) = @_;

      if($bits =  = 0 ) { return 0; }
```

```
    if($bits =  = 1 ) { return 128; }
    if($bits =  = 2 ) { return 192; }
    if($bits =  = 3 ) { return 224; }
    if($bits =  = 4 ) { return 240; }
    if($bits =  = 5 ) { return 248; }
    if($bits =  = 6 ) { return 252; }
    if($bits =  = 7 ) { return 254; }
    if($bits =  = 8 ) { return 255; }
}
```

5.3.3 Discussion

This script performs several different functions. It converts from netmask format to either wildcard or bit count format, from wildcard to either netmask or bit count format, and from bit count to either netmask or wildcard format. Many experienced network engineers pride themselves on doing these conversions in their heads. But it is still relatively common to find router configurations in which the conversion has been done incorrectly.

The difference between netmask and wildcard formats is that netmask format uses ones in the bit pattern to represent bits that do not change, while wildcard format uses zeros to represent these bits. So, for example, if you are constructing an access list that looks at all of the devices in the subnet `192.168.1.0/24`, the netmask would be `255.255.255.0`, and the wildcard in the access list would be `0.0.0.255`.

The reason for the difference is that you will sometimes want to construct an access list that doesn't care which subnet a device is on, but can be used to select a particular set of devices on that subnet. Access lists don't look at subnets; they do pattern matching on addresses.

To convert from wildcard format to netmask format or vice versa, the program simply subtracts each byte in the mask from the number 255, which is 8 bits of all ones. It should be relatively easy to see that this converts all of the ones in a binary pattern to zeros, and all of the zeros to ones.

The conversion to or from CIDR bit count format is slightly more complicated in the program, but easier in concept. If the input is a netmask, the CIDR bit count is simply the number of ones in the bit pattern, counting from the left. Similarly, if the source is a wildcard, the bit count can be found by counting zeros. The program actually has only one subroutine for counting bits. If it needs to convert a wildcard pattern to a bit count, it converts it to netmask format first.

It is important to note that the CIDR bit count format makes sense only if all of the ones in a netmask are on the left, and all of the zeros are on the right. Then the bit count number simply represents the location of the transition from ones to zeros, which in turn represents the division point between the network and host portions of the address. The program includes a check to ensure that the netmask pattern is valid, with no zeros to the left of any ones in the pattern.

Top

Recipe 5.4 Using Static Routing

5.4.1 Problem

You want to configure a static route.

5.4.2 Solution

You can configure a static route with the *ip route* command, as follows:

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#ip route 10.35.15.5 255.255.255.255  Ethernet0
Router(config)#end
Router#
```

You can also configure a static route to point to a particular next-hop router:

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#interface Serial0
Router(config-if)#ip address 10.35.6.2 255.255.255.0
Router(config-if)#exit
Router(config)#ip route 172.16.0.0 255.255.0.0 10.35.6.1 2
Router(config)#end
Router#
```

If you want to ensure that a route remains in place even if the next-hop IP address becomes unreachable or the interface goes down, you can use the *permanent* keyword:

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#ip route 10.35.15.5 255.255.255.255 Ethernet0  permanent
Router(config)#ip route 172.16.0.0 255.255.0.0 10.35.6.1 2  permanent
Router(config)#end
Router#
```

You can also manually configure routing tags using static routes with the *tag* keyword:

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#ip route 172.16.0.0 255.255.0.0 10.35.6.1 2  tag 36291
Router(config)#end
```

```
Router#
```

5.4.3 Discussion

The first version sends all packets destined to the single host `10.35.15.5` out through the `Ethernet0` interface. In this case, the router will need to figure out which device on this segment to forward the packet to, because it must put the MAC address of the next-hop router in the Layer 2 frame header. The standard mechanism for associating IP addresses with MAC addresses is the Address Resolution Protocol (ARP). The router will send out an ARP request broadcast on the Ethernet segment. If the device that owns the packet's destination IP address happens to be on this segment, it will respond with its MAC address. Otherwise, a router configured for proxy ARP will have to respond on its behalf. This is important, because if you do not have proxy ARP configured on the next-hop router, this command will fail. So, for multiple access media such as Ethernet segments, we recommend specifying the IP address of the next-hop router rather than the interface.

Refer to Chapter 22 for more information about enabling and disabling proxy ARP.

You can also specify a point-to-point media such as a serial interface for the route destination:

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#ip route 10.35.15.5 255.255.255.255 Serial0  5
Router(config)#end
Router#
```

In this case there is no ambiguity. You can reach only one other device through this serial interface, so the proxy ARP issues that we just described do not apply.

The *ip route* command in the second example affects any packet whose destination address is in the range from `172.16.0.1` to `172.16.255.254`. These packets will be forwarded to the next-hop router, `10.35.6.1`:

```
Router(config)#ip route 172.16.0.0 255.255.0.0 10.35.6.1 2
```

The last number in this *ip route* command, 2, is the administrative distance for this route. The router will use this distance value to help it to decide between routes to the same destination prefix from different sources. For example, if you have more than one static route to the same destination, or if the router has learned another route to this destination via RIP, it will compare the administrative distances and use the route with the lowest distance value.

If there is no administrative distance value, as in the first example, the router uses a default value of 1.

The syntax for static routes specifies both an IP address and a netmask. This follows the standard rules for netmasks. However, it is useful to remember that the static route statement controls only how packets should be handled on this router. For example, suppose the range `172.16.0.0/16` includes the networks `172.16.1.0/24`, `172.16.2.0/24`, `172.16.5.4/30`, and `172.16.5.8/30`. If all the paths to

all of these networks go through the router whose address is `10.35.6.1`, then they can all be taken together with the same single route statement:

```
Router(config)#ip route 172.16.0.0 255.255.0.0 10.35.6.1 2
```

It is interesting to see what happens when you need to break up a range of addresses. Carrying on with the same example, suppose there is another network, `172.16.3.0/24`, that is connected through a different next-hop router, `10.35.7.2`. You can configure the router as follows:

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#ip route 172.16.0.0 255.255.0.0  10.35.6.1 2
Router(config)#ip route 172.16.3.0 255.255.255.0  10.35.7.2 2
Router(config)#end
Router#
```

This may appear to cause a conflict, because `172.16.3.0/24` is contained within the range `172.16.0.0/16`. However, the longest match rule that we discussed earlier in this chapter resolves the problem. Also note that the router will use the more specific route even if it has a higher administrative distance value. The distance values are used only when selecting between routes with the same mask length. For example, you could configure two static routes to the same destination:

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#ip route 172.16.3.0 255.255.255.0 10.35.6.1  2
Router(config)#ip route 172.16.3.0 255.255.255.0 10.35.7.3  5
Router(config)#end
Router#
```

In this case, as long as the router can reach the better next-hop device (`10.35.6.1`), it will only use this line. The router will install the route with the higher distance only if it doesn't know how to reach the better next-hop device.

Note that this is a cumbersome and unreliable way of achieving automatic rerouting because it works only when the route to the next hop disappears, not when the next hop itself becomes unavailable. So, for example, if these two next-hop routers were connected through different physical interfaces and one of those interfaces went down, the router could switch to the router with the higher distance. But, if both devices were on the same directly connected Ethernet segment, this would not provide a failover. So, while this method is useful for some limited applications, it is generally better to use a dynamic routing protocol such as RIP, EIGRP, or OSPF. These protocols are described in later chapters.

The third example in this recipe uses the *permanent* keyword:

```
Router(config)#ip route 10.35.15.5 255.255.255.255  Ethernet0 permanent
Router(config)#ip route 172.16.0.0 255.255.0.0 10.35.6.1 2  permanent
```

Using the *permanent* keyword ensures that the static route always remains in the routing table, even if

the next-hop interface is down. There is sometimes a danger that the dynamic routing protocol will install a route that you do not want to use, so it may be preferable to drop the packets rather than to use the dynamic route. For example, if you had a private link to another IP network and this link went down, you might not want your routers to try finding a path via the public Internet, even if one were advertised. This method is sort of the opposite of the floating static route given in Recipe 5.5.

The last example in this recipe uses routing tags:

```
Router(config)#ip route 172.16.0.0 255.255.0.0 10.35.6.1 2  tag 36291
```

Routing tags are used when redistributing from one routing protocol to another. They provide a convenient way to tell which routes came from what external protocols or networks. This concept is discussed in more detail in Chapter 6, Chapter 7, Chapter 8, and Chapter 9.

5.4.4 See Also

Recipe 5.5; Chapter 6; Chapter 7; Chapter 8; Chapter 9

Top

Recipe 5.5 Floating Static Routes

5.5.1 Problem

You want to use a static route only when the dynamic route is not available.

5.5.2 Solution

The router will use a floating static route for a particular network prefix only if that same route is not available from the dynamic routing protocol. You can accomplish this by setting the administrative distance of the static route to a value greater than the administrative distance of the dynamic routing protocol:

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#ip route 10.0.0.0 255.0.0.0 172.16.1.1 190
Router(config)#end
Router#
```

You can use the floating static route to trigger a dialer interface:

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#ip route 0.0.0.0 0.0.0.0 Dialer1 190
Router(config)#end
Router#
```

5.5.3 Discussion

The static routes that we discussed in the previous section all had relatively low administrative distance values. If the router has two routes to the same destination but with different distances, it will always choose the one with the lower distance value. This concept also includes the routes that come from other sources, such as dynamic routing protocols.

Every routing protocol has an *administrative distance* that indicates how much the router trusts the information it receives by this method. Table 5-1Table 5-1 shows the default values for these administrative distances. Recipe 5.7 demonstrates how to change these values when they are not appropriate for your network. However, for the current example, the default values are sufficient.

Table 5-3. Cisco default administrative distances

Routing protocol or source	Administrative distance
Connected interface	0
Static route	1
EIGRP summary route	5
External BGP	20
Internal EIGRP	90
IGRP	100
OSPF	110
IS-IS	115
RIP	120
EGP	140
ODR	160
External EIGRP	170
Internal BGP	200
Unknown	255

A floating static route has an administrative distance value that is greater than that of the dynamic routing protocol being used. For example, any route that is learned via OSPF will have a default administrative distance value of 110. This administrative distance applies to all routes learned by this method, regardless of what metric they may have within that protocol.

In the case of static routes, you can directly set the administrative distance for any given static route when you define the route. If the static route's distance value is greater than 110 and OSPF has a route for this destination, the router will use it.

The router will install the floating static route if it doesn't have a similar route from the dynamic routing protocol. Bear in mind, though, that the router will always use the route that has the most precise (longest netmask) match. For example, if the router has learned a route for `10.35.15.0/24` from OSPF, and also has a static route for `10.35.15.0/27`, it will use the static route even if it has a higher administrative distance. The administrative distance is only used to decide between competing routes of the same mask length.

Floating static routes are often used to trigger automated backup mechanisms when the routing protocol

fails. In this case, you could configure a floating static default route, `0.0.0.0/0`, which would point to the dialer interface:

```
Router(config)#ip route 0.0.0.0 0.0.0.0 Dialer1 190
```

If the router loses contact with the rest of the network because of a circuit failure, all of the dynamic routes will drop out of the routing table. The router will then install the floating static route, which will trigger the dial backup connection. We discuss dial backup scenarios in more detail in Chapter 13.

5.5.4 See also

Recipe 5.7; Chapter 13

Top

Recipe 5.6 Using Policy-Based Routing to Route Based on Source Address

5.6.1 Problem

You want to use different network links depending on the source address.

5.6.2 Solution

Policy-based routing allows you to configure special routing rules beyond the normal IP routing table. One common application is to route packets based on the IP source address, rather than (or in addition to) using the destination address:

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#access-list 1 permit 10.15.35.0 0.0.0.255
Router(config)#access-list 2 permit 10.15.36.0 0.0.0.255
Router(config)#interface Ethernet0
Router(config-if)#ip address 10.15.22.7 255.255.255.0
Router(config-if)#ip policy route-map Engineers
Router(config-if)#ip route-cache policy
Router(config-if)#exit
Router(config)#route-map Engineers permit 10
Router(config-route-map)#match ip address 1
Router(config-route-map)#set ip next-hop 10.15.27.1
Router(config-route-map)#exit
Router(config)#route-map Engineers permit 20
Router(config-route-map)#match ip address 2
Router(config-route-map)#set interface Ethernet1
Router(config-route-map)#end
Router#
```

5.6.3 Discussion

This configuration example defines a special routing policy for a group of users defined by the route map called "Engineers." This name is arbitrary, and we recommend that you choose names for your route maps that are meaningful in your organization. This example applies the route map called "Engineers" to all of the packets received on the interface `Ethernet0`. This configuration might be required because certain users need to use a special higher capacity link, for example.

There are two clauses in this route map. The numbers "10" and "20" at the end of each of the *route-map* command lines are used to specify the order in which the router will apply these clauses when making

routing decisions. It is a good practice to use widely spaced numbers like this—it makes it easier to insert new clauses between them. For example, if we needed to add a new clause that had to be executed after the first one but before the last one, we could give it a value of 15.

It is also important to remember that every route map ends with an implicit *deny all*. This ensures that packets that don't match either of the clauses in this route map are unaffected, and will use the standard IP routing table.

The *permit* field is somewhat confusing in general because it governs whether the *route-map* clause will permit or deny the *set* operation contained below it. There are very few cases where the *deny* option is used in practice.

The first route-map clause takes action based on the *match* command:

```
Router(config)#route-map Engineers permit 10
Router(config-route-map)#match ip address 1
Router(config-route-map)#set ip next-hop 10.15.27.1
Router(config-route-map)#exit
```

This command compares the contents of the IP header with access list number 1. Note that this is a standard access list, which is used only to match the source address in the IP header. You can also use extended access lists to match any of the header contents. With extended access lists, you can even match TCP or UDP port numbers. The term *address* in the route-map *match* clause is slightly misleading when extended access lists are used because you can actually match just about anything in the IP header. In this case, however, we are looking for an address.

Access list 1 will match any packet with a source address in the range from `10.15.35.0` to `10.15.35.255`:

```
Router(config)#access-list 1 permit 10.15.35.0 0.0.0.255
```

If the match is successful, the route map will apply the *set* command, which overrides any routing table information and sends the packet to the next-hop router, `10.15.27.1`. Note that this next-hop router must be on a directly connected network because the router will bypass its routing table when it forwards these packets. If the next-hop address is not part of a directly connected subnet, the router can't route the packet because it has already bypassed the routing table in its decision process.

The second clause works in a similar way, but it matches a different access list and sets a different next-hop value:

```
Router(config)#route-map Engineers permit 20
Router(config-route-map)#match ip address 2
Router(config-route-map)#set interface Ethernet1
```

This clause also handles the routing differently. Instead of specifying a next hop, it specifies that any packets matching this rule will be forwarded directly out of the `Ethernet1` interface. This requires that

either the destination device must be on this segment, or a router configured with proxy ARP is available to forward the packet to the ultimate destination.

Because the route map ends with an implicit deny all, all packets that don't match any clause are routed in the normal way using information in the routing table.

You can use the *set next-hop* command to set a series of possible next hops:

```
Router(config-route-map)#set ip next-hop 10.15.27.1 10.15.37.1
```

You can also specify several different possible interfaces:

```
Router(config-route-map)#set interface Ethernet1 Ethernet2
```

This allows you to define what happens if there is no way to route to one of these next-hop devices. If the first address or interface is not available, the router will use the second.

The router will consult its routing table to decide whether the specified next-hop router or interface is available. This is important because, in many cases (such as when using Ethernet segments), it is possible to lose contact with the next-hop router without losing the routing table entry. In this case, the packet will simply be dropped.

In IOS Version 12.0(3)T and later, there is a partial solution to this problem. You can specify the *set ip next-hop verify-availability* command to make the router use CDP to test whether the next-hop device is up:

```
Router(config)#route-map Engineers permit 10
Router(config-route-map)#match ip address 1
Router(config-route-map)#set ip next-hop 10.15.27.1
Router(config-route-map)#set ip next-hop verify-availability
```

Since this uses CDP, you have to ensure that CDP is enabled on the interface leading to this next-hop device. That device must also be running CDP (meaning it must be another Cisco router). This is not a perfect solution because not all media types support CDP, and because the verification process can cause performance problems. Furthermore, CDP uses a relatively long timeout period by default (180 seconds), so it is slow to respond to failures.

If none of the next hops are available, the router simply uses its normal routing table. If the routing table does not have an entry for this route, you may want to specify a default other than the router's general default gateway:

```
Router(config)#route-map Engineers permit 10
Router(config-route-map)#set ip default next-hop 10.15.47.1
```

You can also specify a default interface in a similar way:

```
Router(config)#route-map Engineers permit 10
Router(config-route-map)#set default interface Null0
```

In this case, the default interface is the `Null0` interface. Setting this forces the router to discard the packets rather than using the router's general default gateway.

There is one other extremely important command in the main recipe example, *ip route-cache policy*:

```
Router(config)#interface Ethernet0
Router(config-if)#ip route-cache policy
```

This command tells the router to use fast switching rather than process switching when processing policy commands on the specified interface. This command is new in IOS Version 12.0. Without it, all policy processing is handled by the router's CPU. This can cause serious performance problems in a slower router, or if the traffic load is heavy.

Note, however, that not all policy-based routing commands can be fast switched. In particular, the *set ip default next-hop* and *set default interface* commands are not supported. Furthermore, fast switching has only limited support for the *set interface* command. In particular, the destination interface must have route caching enabled, or it must be a point-to-point link. And, even if these conditions are true, the router still must check its routing table at the process level to ensure that the destination interface is valid. Recipe 5.7 discusses these commands.

Because policy-based routing overrides the normal routing tables within the router, using it can result in some interesting and confusing troubleshooting problems. In the above example, if the next-hop value `10.15.27.1` suffers a failure but its subnet doesn't drop out of the routing table, no dynamic routing protocol is available to find a new path. Worse still, trying to diagnose the problem using normal procedures (such as trying to *ping* from the router to the required destination) will give unpredictable results because the ICMP packets originating on the router will not be subject to the routing policy. You may find that you can *ping*, but that the application doesn't work for certain users.

For this reason, we recommend avoiding policy-based routing unless there is no other way to accomplish the required goals.

5.6.4 See Also

Recipe 5.7

Top

Recipe 5.7 Using Policy-Based Routing to Route Based on Application Type

5.7.1 Problem

You want different applications to use different network links.

5.7.2 Solution

This example is similar to the previous one except that, instead of looking at the source address of the incoming IP packet, it looks at other protocol information such as the TCP or UDP port number. This example redirects HTTP traffic (TCP port 80) from certain source addresses:

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#access-list 101 deny tcp 10.15.25.0 0.0.0.255  any eq www
Router(config)#access-list 101 permit tcp any any eq www
Router(config)#interface Ethernet0
Router(config-if)#ip address 10.15.22.7 255.255.255.0
Router(config-if)#ip policy route-map Websurfers
Router(config-if)#ip route-cache policy
Router(config-if)#exit
Router(config)#route-map Websurfers permit 10
Router(config-route-map)#match ip address 101
Router(config-route-map)#set ip next-hop 10.15.27.1
Router(config-route-map)#exit
Router(config)#route-map Websurfers permit 20
Router(config-route-map)#set ip default next-hop 10.15.26.1
Router(config-route-map)#end
Router#
```

This second example looks at the IP TOS field instead:

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#access-list 102 permit ip any any tos 4
Router(config)#interface Serial0
Router(config-if)#ip address 10.15.23.6 255.255.255.252
Router(config-if)#ip policy route-map High-priority
Router(config-if)#ip route-cache policy
Router(config-if)#exit
Router(config)#route-map High-priority permit 10
Router(config-route-map)#match ip address 102
Router(config-route-map)#set ip next-hop 10.15.27.1
```

```
Router(config-route-map)#end
Router#
```

This third example shows how to use policy-based routing for traffic that originates from the router itself:

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#ip local policy route-map dlswtraffic
Router(config)#access-list 103 permit tcp any any eq 2065
Router(config)#access-list 103 permit tcp any eq 2065 any
Router(config)#route-map dlswtraffic permit 10
Router(config-route-map)#match ip address 103
Router(config-route-map)#set ip next-hop 10.15.27.3
Router(config-route-map)#end
Router#
```

5.7.3 Discussion

These examples show how to route traffic based on protocol information rather than addresses. The first example redirects HTTP packets that originate on any device in the range from `10.15.25.0` to `10.15.25.255`. Access-list 101 has two lines:

```
Router(config)#access-list 101 deny tcp 10.15.25.0 0.0.0.255  any eq www
Router(config)#access-list 101 permit tcp any any eq www
```

The second line selects any TCP packets with any source or destination IP address, and with a destination TCP port number of 80 (HTTP). The first line excludes any packets with the specified range of address and with a destination TCP port number of 80 from the selection.

When the client makes the initial TCP connection, it places a request to the target IP address using a particular port number as the destination. The packet also contains the client's IP address as the source of the packet and specifies a source port number, which is usually a random number greater than 1023.

The first clause in the *route-map* then redirects the traffic matched by this access list to the specified next-hop router:

```
Router(config)#route-map Websurfers permit 10
Router(config-route-map)#match ip address 101
Router(config-route-map)#set ip next-hop 10.15.27.1
```

The second clause in this *route-map* shows how to handle a default next hop:

```
Router(config)#route-map Websurfers permit 20
Router(config-route-map)#set ip default next-hop 10.15.26.1
```

This is invoked as a catchall in case the packet doesn't match the first clause, and there is no appropriate routing table entry that will allow the router to direct it. This clause can be used to prevent

the router from dropping packets with unknown destinations, in case you want to handle them differently (such as sending them to a proxy server). Note that this particular *default next-hop* command specifies the route to be used only if there is no explicit route in the routing table. If there is a route, this clause will not be used.

As we mentioned in Recipe 5.6, using the *set ip default next-hop* command means that the processing of this clause must be done at the process level. Therefore, this type of command can be very CPU-intensive if a large number of packets are involved. If you require this command, it is a good practice to put it at the end of the policy clause list, as we have done here. This way, most of the packets will be handled by one of the previous clauses where they can be fast switched.

The second example shows how to route traffic based on the IP TOS field. Once again, the match is made based on an extended access list:

```
Router(config)#access-list 102 permit ip any any tos 4
```

Please refer to Chapter 11 and Appendix B for more detailed discussions of TOS, IP precedence, and prioritization in general.

The third example shows how to use policy-based routing when traffic originates on the router itself. The router is the source of many types of traffic. This includes several obvious applications such as SNMP network management, Telnet communication with the router's virtual TTY for configuration, and logging. But there are also some less obvious cases where the router is engaged in protocol translation as in Data Link Switching (DLSw) and X.25 over TCP (XOT). This example shows how to use policy-based routing to affect DLSw packets that originate with this router. Chapter 15 discusses DLSw in more detail.

The only important difference between local policy-based routing and the earlier examples (which were tied to particular interfaces) is the global configuration command *ip local policy route-map*.

```
Router(config)#ip local policy route-map dlswtraffic
```

This command applies the *dlswtraffic* policy to all locally generated traffic.

5.7.4 See Also

Recipe 5.5; Chapter 11; Chapter 15; Appendix B

Top

Recipe 5.8 Examining Policy-Based Routing

5.8.1 Problem

You want to see information about how policy-based routing has been applied on a router.

5.8.2 Solution

The *show ip policy* command shows what routing policies have been applied on a router. Here is the output for a router that has all three of the policies from Recipe 5.7:

```
Router>show ip policy
Interface      Route map
local          dlswtraffic
Ethernet0      Websurfers
Serial0        High-priority
```

You can see more detail on what each of these policies does by looking at the route maps:

```
Router>show route-map
route-map High-priority, permit, sequence 10
  Match clauses:
    ip address (access-lists): 101
  Set clauses:
    ip next-hop 10.15.27.1
  Policy routing matches: 0 packets, 0 bytes
route-map Websurfers, permit, sequence 10
  Match clauses:
    ip address (access-lists): 102
  Set clauses:
    ip next-hop 10.15.27.1
  Policy routing matches: 0 packets, 0 bytes
route-map Websurfers, permit, sequence 20
  Match clauses:
  Set clauses:
    ip default next-hop 10.15.26.1
  Policy routing matches: 4 packets, 531 bytes
route-map dlswtraffic, permit, sequence 10
  Match clauses:
    ip address (access-lists): 103
  Set clauses:
    ip next-hop 10.15.27.3
  Policy routing matches: 5 packets, 500 bytes
```

5.8.3 Discussion

The first command, *show ip policy*, tells you about all of the routing policies that have been applied on the router. The second command, *show route-map*, shows all of the route maps. It is important to note that the first command shows only the routing policies that have actually been applied on the router, whether for local traffic or packets coming from an interface. It also shows all applied routing policies, whether the interfaces involved are active or not. The second command shows all configured route maps, whether they had been applied to anything or not.

The *show route-map* command also gives useful information about how the route maps are being used. Notice that the second clause of the *Websurfers* route map has matched 4 packets for a total of 531 bytes since it was applied, and the *dlswtraffic* route map has similarly matched 5 packets for a total of 500 bytes. You can dig a little further by looking at the access lists that these route maps use to match packets:

```
router>show access-list 103
Extended IP access list 103
    permit tcp any any eq 2065 (3 matches)
    permit tcp any eq 2065 any (2 matches)
```

This shows not only the details of how the access list used in the route map works, but it also precisely tells which lines are being used.

5.8.4 See Also

Recipe 5.7

Top

Recipe 5.9 Changing Administrative Distances

5.9.1 Problem

You want to change the administrative distance for an external network.

5.9.2 Solution

Use the *distance* command to adjust the administrative distance for a particular routing protocol. The precise syntax depends on the routing protocol. This example uses RIP:

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#router rip
Router(config-route)#network 192.168.15.0
Router(config-route)#distance 15 192.168.15.1 0.0.0.0
Router(config-route)#distance 200 192.168.15.0 0.0.0.255
Router(config-route)#distance 255
Router(config-route)#end
Router#
```

For EIGRP, you can specify the distance for routes learned from both internal and external neighbors:

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#router eigrp 111
Router(config-route)#network 192.168.16.0
Router(config-route)#distance eigrp 55 200
Router(config-route)#end
Router#
```

With OSPF, you can also control the distance depending on whether the neighboring router is in the same area:

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#router ospf 66
Router(config-route)#distance ospf inter-area 115
Router(config-route)#distance ospf intra-area 105
Router(config-route)#distance ospf external 125
Router(config-route)#end
Router#
```

Finally, you can configure BGP distances for internal, external, and local routes:

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#router bgp 65520
Router(config-route)#distance bgp 115 220 50
Router(config-route)#end
Router#
```

5.9.3 Discussion

When dealing with administrative distances, always bear in mind that they only affect route selection on a particular router. Administrative distances are not distributed by the routing protocols. However, they can affect the routing protocols; in particular, distance vector algorithms distribute only locally used routes. Adjusting the administrative distance could change which routes are used, so you should always be extremely careful when changing them.

The main place where it makes sense to consider changing the administrative distance for a routing protocol is when the same route is distributed by two different protocols. For example, suppose you have an OSPF network that uses EIGRP to distribute routes for dial backup links. When both the primary OSPF link and backup EIGRP link are active, the router will prefer the routes it learns via EIGRP by default (because of the lower administrative distance values). This will cause performance problems, since the backup link most likely has lower bandwidth and it may also interfere with dropping the dialup connection after the primary link recovers because the backup is actively passing traffic. Changing the administrative distance can be an effective way to resolve this kind of conflict. However, we want to stress that this is a relatively unusual case.

The default values for administrative distances are given in Table 5-1 and discussed in Recipe 5.5. You can adjust these values to force the router to prefer routes learned by one protocol over those learned from another. In Recipe 5.5, we gave high administrative distances to static routes so that the router would not use them unless the dynamic routing protocol was unavailable. This recipe shows that it is also possible to modify the administrative distances for the routing protocols themselves to allow one protocol to dominate over another.

In the RIP example shown in the solution section, the last *distance* command overrides the default administrative distance for RIP routes and sets it to 255:

```
Router(config)#router rip
Router(config-route)#distance 255
```

This is the worst possible administrative distance value. Any route with an administrative distance of 255 is considered infinitely far away and will never be used. You could also configure a lower administrative distance value for this protocol. For example, if we gave it a value of 105, the RIP routes would be better than OSPF but worse than IGRP by default.

The first *distance* command line overrides the new default value for routes received from one particular router, `192.168.15.1`. These routes will have a distance value of 15.

```
Router(config-route)#distance 15 192.168.15.1 0.0.0.0
```

The last field in this line is an address wildcard pattern. In the binary representation of this number, the zeros represent values in the address that are important and ones represent values that can be ignored. In this case, the wildcard is `0.0.0.0`, which means that every bit is important. So this refers to a single specific device, which is a router whose RIP routes we want to treat differently.

The other *distance* command line in the RIP example changes the administrative distance value to 200 for routes learned from devices in the `192.168.15.0` subnet:

```
Router(config-route)#distance 200 192.168.15.0 0.0.0.255
```

Because last field is `0.0.0.255`, this command includes every device in the `192.168.15.0/24` subnet.

The router processes these commands sequentially and stops as soon as it gets a match. So, although the device `192.168.15.1` is also a member of the `192.168.15.0` subnet, its routes receive an administrative distance of 15. However, if the two commands were set in the opposite order, the more specific one would never be invoked.

This example represents another interesting use for administrative distances. You can use them (as we have in this RIP example) to ensure that you favor routes learned from particular routers over other less reliable devices. Note that this is only possible with distance vector protocols, so it will not work, for example, with OSPF. Refer to Chapter 6 for more information about RIP.

The EIGRP and OSPF configurations are similar, except that they include different options. The EIGRP example uses the *distance eigrp* command:

```
Router(config)#router eigrp 111
Router(config-route)#distance eigrp 55 200
```

This sets the administrative distance for routes learned by EIGRP to 55 if they are from the same EIGRP network, or 200 if they are external routes that have been redistributed into EIGRP.

OSPF also allows you to set a different administrative distance for external routes. And, for internal routes, you can specify different administrative distances depending on whether the route destination is internal or external to the area. This might be useful in certain cases where you want to connect to another area through an external network rather than through the OSPF core. However, in most cases you do not need to treat intra-area and inter-area OSPF routes differently.

When setting distances for use with BGP, the syntax is *distance bgp <external> <internal> <local>*.

```
Router(config-route)#distance bgp 115 220 50
```

The external and internal distances are used for routes learned through the eBGP and iBGP protocols,

respectively. Local routes in this context are routes to destinations inside of this Autonomous System (AS). For more information on BGP, please refer to Chapter 9. Adjusting BGP administrative distances is relatively rare and frequently dangerous. The default values of 20 for eBGP and 200 for iBGP means that the router will generally prefer external BGP routes, but will prefer any internal routing protocol to iBGP. This is almost always the required behavior. If you want to change the way the router handles BGP routes, we don't recommend using administrative distance. We discuss several other options for manipulating BGP routes in Chapter 9.

5.9.4 See Also

Recipe 5.4; Chapter 6; Chapter 7; Chapter 8; Chapter 9

Top

Recipe 5.10 Routing Over Multiple Paths with Equal Costs

5.10.1 Problem

You want to restrict how many paths your router can use simultaneously to reach a particular destination.

5.10.2 Solution

By default, the router will install up to four routes to the same destination for most routing protocols, except for BGP (where the default is one), and static routes (which allow six). You can change this default to any value between one and six by using the *maximum-paths* configuration command:

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#router ospf 65510
Router(config-router)#maximum-paths 2
Router(config-router)#end
Router#
```

The same syntax works for other routing protocols:

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#router eigrp 99
Router(config-router)#maximum-paths 2
Router(config-router)#end
Router#
```

In IOS Version 12.2T, Cisco introduced a new command to allow you to configure the number of iBGP paths separately from eBGP. You set the maximum number of eBGP paths using the standard *maximum-paths* command, and specify the number of paths for iBGP separately:

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#router bgp 65511
Router(config-router)#maximum-paths 2
Router(config-router)#maximum-paths ibgp 3
Router(config-router)#end
Router#
```

IOS releases prior to 12.2T include only the standard *maximum-paths* command. This command sets the

maximum paths for both iBGP and eBGP.

5.10.3 Discussion

In large, highly redundant networks, it is common to have many possible paths between two points. By default, the router will install up to four routes to the same destination in the routing table for most routing protocols, provided that all of these routes have the same cost. But sometimes this is either too many or too few, so these commands offer a way to change the defaults.

For BGP the default value is one path because of the sheer number of routes in the public Internet and the desire to simplify things where possible. However, BGP is not always used to connect to the Internet. Even when it is, there are times when you want to share the load between two Internet connections. With IOS Version 12.2T, Cisco has introduced the ability to control the number of iBGP paths separately from the number of eBGP paths using the command:

```
Router(config)#router bgp 65511
Router(config-router)#maximum-paths ibgp 2
```

This is useful in situations where there are several Autonomous System Boundary Routers (ASBRs) sharing routing information within a complex, multiply connected Autonomous System, but you still wish to keep your external routing tables as simple as possible.

Without the *ibgp* keyword, this command affects both iBGP and eBGP. So, if you want two different values, you can specify them separately, as we have shown. Before IOS Version 12.2T, you could specify only a single value for both iBGP and eBGP. For BGP, the default value for this parameter is one. However, for all other protocols, the default is four.

When there are several paths available in the routing table, the router will generally alternate between the different possible paths. When you use process switching, the router will alternate packets among the different paths. With fast switching, however, it will alternate *flows*, rather than individual packets. For TCP connections, this means that each separate connection will find one of the available paths and use this path exclusively. For UDP and ICMP, however, every packet is a separate flow, so successive packets will likely take different paths.

Alternating flows rather than packets results in slightly less efficient load sharing of TCP traffic, but it has the benefit of causing packets to always arrive in the same order that they were sent. When packets take different paths through the network, they can arrive out of sequence. So, in general, it is better to use per-flow rather than per-packet load sharing.

By default, Cisco Express Forwarding (CEF) will also load balance by alternating flows rather than individual packets. You can configure CEF to load balance on a packet-by-packet basis using the interface configuration command *ip load-sharing per-packet*:

```
Router#configure terminal
```

```
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#ip cef
Router(config)#interface Ethernet0
Router(config-if)#ip load-sharing per-packet
Router(config-if)#end
Router#
```

We discuss CEF in more detail in Chapter 11.

Top

Chapter 6. RIP

Top

Introduction

RIP Version 1 was the Internet's first widely used routing protocol. It was standardized in RFC 1058, although implementations of the protocol based on de facto standards existed much earlier. It is still useful in small, simple networks. RIP Version 2 is documented in RFC 1723. All Cisco routers support RIP Version 1. Version 2 support was integrated into IOS Version 11.1. A detailed discussion of RIP Versions 1 and 2 is beyond the scope of this book. If you are unfamiliar with dynamic routing protocols in general or with RIP in particular, you can find theoretical descriptions of how the protocol works in IP Routing (O'Reilly) and Designing Large-Scale LANs (O'Reilly). We also recommend reading the appropriate RFCs.

RIP is useful in some situations, but you have to remember its limitations. First, it is a purely classful protocol, and Version 1 doesn't support variable length subnet masks. So you should not use this protocol if you do any complex subnetting. Second, both Versions 1 and 2 of RIP use the very small metric value of 16 to signify "infinity." The protocol considers any network that is more than 16 hops away to be unreachable. This is particularly important if you adjust any metric values to make RIP favor a fast link over a slow one. In practice, it is quite easy to exceed the maximum metric, even in small networks.

However, RIP can be extremely useful over small parts of a network. For example, it is much easier to configure than BGP as a method for interconnecting two or more different OSPF or EIGRP Autonomous Systems. And, because RIP has been around for so long, it is often useful when exchanging routing information with legacy equipment. Indeed, it is almost impossible to find a router of any age from any vendor that doesn't implement RIP.

In this book, we assume that you are familiar with RIP in general and focus on Cisco's implementation of it. We also discuss some specific issues that we think are particularly important.

One of the central features of RIP is that it distributes the entire routing table every 30 seconds. The protocol requires every device to add a small random amount to this time period, but doesn't specify the size of this offset, or whether it should be positive or negative. Cisco routers always reduce this period slightly by subtracting a random variable amount of time, up to 4.5 seconds. This helps to prevent the synchronization problems caused by several routers sending their updates simultaneously, which can in turn cause network loading problems.

If a particular route is not seen for 6 successive update cycles, or 180 seconds by default, the routers will mark it as invalid. They will then flush the invalid route from their routing tables if they don't see it for 8 cycles, or 240 seconds. This implies that RIP is slow to converge after a failure, but the network will actually converge much more quickly if it can take advantage of a protocol feature called triggered updates. This means that when a route's metric suddenly changes, for whatever reason, a router using

triggered updates will not wait for the full update cycle before distributing information about the change to the other routers in the network.

This is different from the modification to RIP described in Recipe 6.11, which is also called a triggered update. This feature, which is a partial implementation of RFC 2091, allows the routers to send routing updates only when there are changes, instead of sending the entire routing table at each update cycle. This makes it possible to configure the routers to just send incremental changes. Using triggered updates drastically improves RIP performance, but it must be configured on all of the routers sharing the link. It is often unsupported by legacy equipment, and Cisco routers support this feature only on point-to-point serial links.

Cisco routers implement a hold-down timer with RIP. This is a protocol feature that is not described in the standard protocol RFCs. When the router marks a route invalid, it starts the hold-down timer, which is 180 seconds by default, and ignores all updates for this route. This helps to make the network somewhat more stable.

Because RIP uses a distance vector algorithm rather than a link state protocol (like OSPF), you can use Cisco's distribute lists to make a router distribute only certain routes. This allows you to prevent distribution of routing information that you don't want to be generally visible. And you can also reject incoming routing information that you don't want to use. This can be extremely useful when exchanging routing information between networks, or when connecting small networks' regions with legacy equipment.

Top

Recipe 6.1 Configuring RIP Version 1

6.1.1 Problem

You want to run RIP on a simple network.

6.1.2 Solution

The following commands show how to configure basic RIP functionality:

```
Router2#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router2(config)#interface Ethernet0
Router2(config-if)#ip address 192.168.30.1 255.255.255.0
Router2(config-if)#interface Serial0.1
Router2(config-subif)#ip address 172.25.2.2 255.255.255.0
Router2(config-subif)#exit
Router2(config-router)# router rip
Router2(config-router)#network 172.25.0.0
Router2(config-router)#network 192.168.30.0
Router2(config-router)#end
Router2#
```

6.1.3 Discussion

You enable RIP for an interface by associating its IP address with a network. For example, the `Serial0.1` interface in this example has an IP address of `172.25.2.2`. So, if you want this interface to take part in RIP route distribution, you just need to include a network statement that includes its address:

```
Router2(config)#router rip
Router2(config-router)#network 172.25.0.0
```

If you have other interfaces that are also part of `172.25.0.0/16` on this router, they will take part in RIP as well. It's important to note that RIP network statements work with classful network addresses. Since `172.25.0.0` is a Class B network, the statement *network 172.25.0.0* tells the router to include all interfaces in this range. Note, however, that the other address on this router, `192.168.30.1`, is part of the Class C network `192.168.30.0/24`, which is why we need the second *network* command. If your router has several addresses that belong to different network classes (for example, `192.168.x.0/24`), you have to specify them all separately:

```
Router2(config)#router rip
Router2(config-router)#network 192.168.1.0
Router2(config-router)#network 192.168.2.0
Router2(config-router)#network 192.168.3.0
```

It can be somewhat confusing to know which networks to specify. If your router receives a route from another router, it will pass it along to other routers whether you have specified a network statement for this address or not. However, the router will only distribute information about directly connected routes if there is a network statement for these routes. And the router will exchange RIP information only through interfaces specified by *network* statements.

The *show ip protocol* command gives a lot of useful information about what interfaces and address ranges are involved in the routing protocol:

```
Router2#show ip protocol
Routing Protocol is "rip"
  Sending updates every 30 seconds, next due in 7 seconds
  Invalid after 180 seconds, hold down 0, flushed after 240
  Outgoing update filter list for all interfaces is not set
  Incoming update filter list for all interfaces is not set
  Redistributing: rip
  Default version control: send version 1, receive any version
    Interface             Send  Recv  Triggered RIP  Key-chain
     Ethernet0            1     1 2
     Serial0.1            1     1 2
  Automatic network summarization is in effect
  Maximum path: 4
  Routing for Networks:
    172.25.0.0
    192.168.30.0
  Routing Information Sources:
    Gateway         Distance      Last Update
    172.25.2.1           120      00:00:17
  Distance: (default is 120)

Router2#
```

As you can see from the output of this command, the router will send RIP Version 1 on both interfaces `Ethernet0` and `Serial0.1`, but it will listen for both Versions 1 and 2. You can force the router to use only Version 1 to send and receive routing information for all interfaces with the following command:

```
Router2#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router2(config)#router rip
Router2(config-router)#version 1
Router2(config-router)#end
Router2#
```

However, it is usually better to restrict what you send rather than what you receive. That way, if another

device is misconfigured to send the other version, it doesn't break the network. We will discuss how to configure different RIP version options in Recipe 6.13.

Once you have RIP running successfully between two or more routers, you can look at your routing table to see which routes the router is learning and from where it is learning them:

```
Router2#show ip route rip
R    172.22.0.0/16 [120/1] via 172.25.2.1, 00:00:00, Serial0.1
R    172.25.1.0/24 [120/1] via 172.25.2.1, 00:00:00, Serial0.1
R    192.168.20.0/24 [120/2] via 172.25.2.1, 00:00:00, Serial0.1
Router2#
```

This command just gives a subset of the output of the more general *show ip route* command discussed in Chapter 5. The example output shows that the router has learned that it can get to the destination network `192.168.20.0/24` through the next-hop router, `172.25.2.1`, which is on interface `Serial0.1`.

The numbers in the square brackets, `[120/2]`, are the administrative distance and the RIP metric, respectively. Cisco routers give all RIP routes a default administrative distance of 120. The metric value of 2 indicates the distance to this network. By default, the metric is the same as the number of hops, but you can adjust metric values to break this rule. We discuss how to change metric values in Recipe 6.3, Recipe 6.4, Recipe 6.5, and Recipe 6.8, and we talk about administrative distances in Chapter 5.

Another useful command for looking at RIP functions on your router is the *show ip rip database* command, introduced in IOS 12.0(6)T:

```
Router2#show ip rip database
172.22.0.0/16    auto-summary
172.22.0.0/16
    [1] via 172.25.2.1, 00:00:13, Serial0.1
172.25.0.0/16    auto-summary
172.25.1.0/24
    [1] via 172.25.2.1, 00:00:13, Serial0.1
172.25.2.0/24    directly connected, Serial0.1
192.168.20.0/24    auto-summary
192.168.20.0/24
    [2] via 172.25.2.1, 00:00:13, Serial0.1
192.168.30.0/24    auto-summary
192.168.30.0/24    directly connected, Ethernet0
Router2#
```

This allows you to see more detailed information about each of the routes in the RIP database.

6.1.4 See Also

Recipe 6.3; Recipe 6.4; Recipe 6.5; Recipe 6.8; Recipe 6.13; Chapter 5

Top

Recipe 6.2 Filtering Routes with RIP

6.2.1 Problem

You want to restrict what routing information is exchanged within RIP.

6.2.2 Solution

You can filter inbound RIP routes on a per-interface basis with a distribute list:

```
Router2#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router2(config)#access-list 10 deny 192.168.20.0
Router2(config)#access-list 10 permit any
Router2(config)#router rip
Router2(config-router)#distribute-list 10 in Serial0.1
Router2(config-router)#network 172.25.0.0
Router2(config-router)#network 192.168.30.0
Router2(config-router)#end
Router2#
```

This configuration example shows how to filter outbound RIP-based routes on a per-interface basis:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#access-list 20 permit 0.0.0.0
Router1(config)#access-list 20 deny any
Router1(config)#router rip
Router1(config-router)#distribute-list 20 out Serial0/0.2
Router1(config-router)#  network 172.25.0.0
Router1(config-router)#end
Router1#
```

6.2.3 Discussion

The access list in the first configuration example in this recipe prevents this router from accepting any routing information about the network `192.168.20.0`. You can see that this route, which was visible in Recipe 6.1, no longer appears in the routing table:

```
Router2#show ip route rip
R    172.22.0.0/16 [120/1] via 172.25.2.1, 00:00:21, Serial0.1
R    172.25.1.0/24 [120/1] via 172.25.2.1, 00:00:21, Serial0.1
```

```
Router2#
```

The *show ip protocol* command shows which interfaces have inbound or outbound distribute lists:

```
Router2#show ip protocol
Routing Protocol is "rip"
  Sending updates every 30 seconds, next due in 27 seconds
  Invalid after 180 seconds, hold down 0, flushed after 240
  Outgoing update filter list for all interfaces is not set
  Incoming update filter list for all interfaces is not set
    Serial0.1 filtered by 10 (per-user), default is 10
  Redistributing: rip
  Default version control: send version 1, receive any version
    Interface             Send  Recv  Triggered RIP  Key-chain
    Ethernet0             1     1 2
    Loopback0             1     1 2
    Serial0.1             1     1 2
  Automatic network summarization is in effect
  Maximum path: 4
  Routing for Networks:
    172.25.0.0
    192.168.30.0
  Routing Information Sources:
    Gateway         Distance      Last Update
    172.25.2.1           120      00:00:17
  Distance: (default is 120)

Router2#
```

This shows that the interface `Serial0.1` uses access-list 10 to filter incoming routing information. You can then use the *show access-list* command to see what this affects.

If you control both the sending and receiving routers, it is usually best to filter the routes before sending them instead of sending them across the network and then ignoring them. So inbound filtering is most common in situations where you are receiving routes from a device that you don't control. Since RIP frequently runs on end devices such as Unix servers, inbound filtering is fairly common.

You can use outbound filtering, on the other hand, for reducing the size of routing tables on access routers. For example, it is extremely useful in hub-and-spoke type WANs. In this case, each remote branch router cares only about its local segments and "everything else." It can reach all of the non-local routes via the hub router, so you can reduce unnecessary WAN bandwidth utilization and memory consumption on the branch router by configuring the hub router to send out only a single default route. In fact, when used in conjunction with the non-periodic update feature discussed in Recipe 6.11, this makes a good WAN routing solution.

The second example in the solution section of this recipe shows the configuration of a hub router so that it sends only the default route, `0.0.0.0/0`. The routing table of the other router then becomes extremely simple:

```
Router2#show ip route rip
R*    0.0.0.0/0 [120/5] via 172.25.2.1, 00:00:02, Serial0.1
Router2#
```

The *show ip protocol* command shows the filter on the hub router:

```
Router1#show ip protocol
Routing Protocol is "rip"
  Sending updates every 30 seconds, next due in 9 seconds
  Invalid after 180 seconds, hold down 180, flushed after 240
  Outgoing update filter list for all interfaces is not set
    Serial0/0.2 filtered by 20 (per-user) , default is 20
  Incoming update filter list for all interfaces is not set
  Redistributing: rip
  Default version control: send version 1, receive any version
    Interface             Send  Recv  Triggered RIP  Key-chain
    FastEthernet0/0.1     1     1 2
    Serial0/0.2           1     1 2
    FastEthernet0/1       1     1 2
   Automatic network summarization is in effect
  Maximum path: 4
  Routing for Networks:
    172.22.0.0
    172.25.0.0
  Routing Information Sources:
    Gateway         Distance      Last Update
    172.25.1.7           120      00:00:23
    172.25.2.2           120      00:00:07
    172.22.1.4           120      00:00:19
  Distance: (default is 120)

Router1#
```

You can also configure the router to filter all interfaces simultaneously with a single rule:

```
Router2#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router2(config)#access-list 10 deny 192.168.20.0
Router2(config)#access-list 10 permit any
Router2(config)#router rip
Router2(config-router)#distribute-list 10 in
Router2(config-router)#end
Router2#
```

This feature is rarely used, because you usually want apply different filters to different interfaces depending on what other devices are connected. But, when you want to explicitly eliminate certain unwanted routes from your network, regardless of where they originate, this is the easiest way to do it. With the *show ip protocols* command you can see that access list number 10 has been applied to traffic coming in from all interfaces:

```
Router2#show ip protocols
```

```
Routing Protocol is "rip"
  Sending updates every 30 seconds, next due in 0 seconds
  Invalid after 180 seconds, hold down 0, flushed after 240
  Outgoing update filter list for all interfaces is not set
  Incoming update filter list for all interfaces is 10
  Redistributing: rip
  Default version control: send version 1, receive any version
    Interface             Send  Recv  Triggered RIP  Key-chain
    Ethernet0             1     1 2
    Loopback0             1     1 2
    Serial0.1             1     1 2
  Automatic network summarization is in effect
  Maximum path: 4
  Routing for Networks:
    172.25.0.0
    192.168.30.0
  Routing Information Sources:
    Gateway         Distance      Last Update
    172.25.2.1           120      00:00:03
  Distance: (default is 120)

Router2#
```

It's important to remember that while you can use global distribute lists together with interface-specific distribute lists, the result actually combines the effects of both. If you have a global distribute list that blocks a particular network, and an interface list that blocks another address, the router will block both addresses on that interface.

6.2.4 See Also

Recipe 6.11

Top

Recipe 6.3 Redistributing Static Routes into RIP

6.3.1 Problem

You want RIP to redistribute static routes that you have configured on your router.

6.3.2 Solution

The *redistribute static* command tells RIP to forward static routes in addition to the directly connected routes and the routes that have been learned from other RIP routers, which it forwards by default:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#ip route 192.168.10.0 255.255.255.0 172.22.1.4
Router1(config)#router rip
Router1(config-router)#redistribute static
Router1(config-router)#end
Router1#
```

You can define how these routes look to other routers when they are redistributed:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#ip route 192.168.10.0 255.255.255.0 172.22.1.4
Router1(config)#router rip
Router1(config-router)#redistribute static metric 5
Router1(config-router)#distribute-list 7 out static
Router1(config-router)#exit
Router1(config)#access-list 7 permit 192.168.10.0
Router1(config)#end
Router1#
```

6.3.3 Discussion

The biggest potential problem that you will encounter with redistributing routes into RIP comes from breaking network class boundaries. RIP is classful, so you have to be rather careful about how you distribute routing information from other sources that may be classless. In this recipe, Router1 redistributes a static route for the Class C network `192.168.10.0`. But if we tried instead to redistribute a larger range (such as `192.168.12.0/22`), RIP would not generate any errors—the router would just quietly refuse to forward this route.

Looking at the RIP database on a router with IOS level 12.0(6)T or higher shows the redistributed static route:

```
Router1#show ip rip database 192.168.10.0 255.255.255.0
192.168.10.0/24    redistributed
    [5] via 0.0.0.0,
Router1#
```

After configuring the second example, the output of *show ip protocols* includes information about the filtering. This command also tells you what protocols RIP is distributing:

```
Router1#show ip protocols
Routing Protocol is "rip"
  Sending updates every 30 seconds, next due in 5 seconds
  Invalid after 180 seconds, hold down 180, flushed after 240
  Outgoing update filter list for all interfaces is not set
    Redistributed static filtered by 7
  Incoming update filter list for all interfaces is not set
  Redistributing: static, rip
  Default version control: send version 2, receive version 2
    Interface             Send  Recv  Triggered RIP  Key-chain
    FastEthernet0/0.1     2     2
    Serial0/0.2           2     2
    FastEthernet0/1       2     2
   Automatic network summarization is in effect
  Maximum path: 4
  Routing for Networks:
    172.22.0.0
    172.25.0.0
  Routing Information Sources:
    Gateway         Distance      Last Update
    172.25.1.7           120      00:00:03
    172.25.2.2           120      00:00:06
    172.22.1.4           120      00:00:08
  Distance: (default is 120)
Router1#
```

In addition to static routes, you can distribute information from other dynamic routing protocols with RIP simply by specifying which protocol's routes you want RIP to use. For example, if you have an EIGRP network that uses process number `65530` on the same router, you would redistribute the EIGRP routes into RIP like this:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#router eigrp 65530
Router1(config-router)#network 192.168.1.0
Router1(config-router)#exit
Router1(config)#router rip
Router1(config-router)#redistribute eigrp 65530
Router1(config-router)#end
```

```
Router1#
```

If you look at the *show ip protocols* command, you can see that RIP redistributes routes it learns from EIGRP, but EIGRP does not redistribute routes learned from RIP. If you also want EIGRP to redistribute RIP routes, you must explicitly configure it to do so. We discuss EIGRP configuration and offer redistribution examples in Chapter 7.

```
Router1#show ip protocols
Routing Protocol is "rip"
  Sending updates every 30 seconds, next due in 0 seconds
  Invalid after 180 seconds, hold down 180, flushed after 240
  Outgoing update filter list for all interfaces is
  Incoming update filter list for all interfaces is
  Redistributing: static, rip, eigrp 65530


  Default version control: send version 1, receive any version
    Interface         Send  Recv   Key-chain
    FastEthernet0/0.1     2     2
    Serial0/0.2           2     2
    FastEthernet0/1       2     2
   Automatic network summarization is in effect
  Maximum path: 4
  Routing for Networks:
    172.22.0.0
    172.25.0.0
  Routing Information Sources:
    Gateway         Distance      Last Update
    172.25.1.7           120      00:00:03
    172.25.2.2           120      00:00:06
    172.22.1.4           120      00:00:08
  Distance: (default is 120)

Routing Protocol is "eigrp 65530"
  Outgoing update filter list for all interfaces is
  Incoming update filter list for all interfaces is
  Default networks flagged in outgoing updates
  Default networks accepted from incoming updates
  EIGRP metric weight K1=1, K2=0, K3=1, K4=0, K5=0
  EIGRP maximum hopcount 100
  EIGRP maximum metric variance 1
  Redistributing: eigrp 65530
  Automatic network summarization is in effect
  Routing for Networks:
    192.168.1.0
  Routing Information Sources:
    Gateway         Distance      Last Update
  Distance: internal 90 external 170

Router1#
```

Table 6-1 shows a list of foreign protocols that RIP can redistribute.

Table 6-1. Protocols that RIP can redistribute

Type	Description
BGP	Border Gateway Protocol
Connected	Connect interfaces
EGP	Exterior Gateway Protocol
EIGRP	Enhanced IGRP
IGRP	Interior Gateway Routing Protocol
ISIS	ISO IS-IS Routing Protocol
Mobile	Mobile routes
OSPF	Open Shortest Path First
RIP	Routing Information Protocol
Static	Static routes

6.3.4 See Also

Recipe 6.4; Chapter 7

Top

Recipe 6.4 Redistributing Routes Using Route Maps

6.4.1 Problem

You want to use route maps for more detailed control over how RIP redistributes routing information from other sources.

6.4.2 Solution

Route maps give you much better control over how RIP redistributes external routes. This example uses static routes, but the same principles apply when redistributing routes from other protocols:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#ip route 192.168.10.0 255.255.255.0 172.22.1.4
Router1(config)#ip route 192.168.11.0 255.255.255.0 172.22.1.4
Router1(config)#ip route 192.168.12.0 255.255.255.0 172.22.1.4
Router1(config)#access-list 20 permit 192.168.10.0
Router1(config)#access-list 21 permit 192.168.11.0
Router1(config)#route-map STATIC permit 10
Router1(config-route-map)#  match ip address 20
Router1(config-route-map)#  set metric 2
Router1(config-route-map)#  set tag 2
Router1(config-route-map)#!
Router1(config-route-map)#route-map STATIC permit 20
Router1(config-route-map)#  match ip address 21
Router1(config-route-map)#  set metric 8
Router1(config-route-map)#!
Router1(config-route-map)#route-map STATIC deny 30
Router1(config-route-map)#exit
Router1(config)#router rip
Router1(config-router)#redistribute static route-map STATIC
Router1(config-router)#end
Router1#
```

6.4.3 Discussion

In this example, we want RIP to distinguish between the different routes. There are two parameters that we can change, the metric and the route tag. We have already discussed metrics, which basically represent the distance or cost of the path to the remote network. A route tag is simply an arbitrary number that RIP will attach to a particular route. RIP doesn't actually use these tag values, but as we

discuss in Recipe 6.16, RIP Version 2 will distribute them with the routing table so that you can use this information when distributing these routes into other routing protocols.

The route map in this recipe has three clauses. The first assigns a metric of 2 and a tag of 2 to the network `192.168.10.0`. Then it gives a metric of 8 to`192.168.11.0`. The last clause discards all other routes. This is different from Recipe 6.3, in which all routes were redistributed with the same metric.

This kind of redistribution can be useful when you want to control how traffic load is balanced between two links. For example, if you have two routers that both connect to the same set of remote networks, you could balance them so that one router has a better metric for half of the routes, and the other has a better metric for the rest of the routes. In this way you can ensure that both links are used equally, and if one router fails, the other will take over for it.

It is often easier to understand route maps by looking at the *show route-map* command, rather than the configuration commands. In this case, for example, you can see exactly what access lists apply to each clause, along with all of the values that are set as a result. This output also shows how many times each policy has been applied:

```
Router1#show route-map STATIC
route-map STATIC, permit, sequence 10
  Match clauses:
    ip address (access-lists): 20
  Set clauses:
    metric 2
    tag 2
  Policy routing matches: 0 packets, 0 bytes
route-map STATIC, permit, sequence 20
  Match clauses:
    ip address (access-lists): 21
  Set clauses:
    metric 8
   Policy routing matches: 0 packets, 0 bytes
route-map STATIC, deny, sequence 30
  Match clauses:
  Set clauses:
  Policy routing matches: 0 packets, 0 bytes
Router1#
```

Looking at the RIP database, you can see the metrics that RIP uses for distributing each of these static routes:

```
Router1#show ip rip database
192.168.10.0/24    auto-summary
192.168.10.0/24    redistributed
    [2] via 0.0.0.0,
192.168.11.0/24    auto-summary
192.168.11.0/24    redistributed
    [8] via 0.0.0.0,
Router1#
```

You can also use the route map technique for distributing routes from dynamic routing protocols. For example, if you had another route map called *EIGRPMAP* that you wanted to use when redistributing routes learned through EIGRP process number `65530`, you could configure the router like this:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#router eigrp 65530
Router1(config-router)#network 192.168.1.0
Router1(config-router)#exit
Router1(config)#router rip
Router1(config-router)#redistribute eigrp 65530 route-map EIGRPMAP
Router1(config-router)#end
Router1#
```

6.4.4 See Also

Recipe 6.3; Recipe 6.16

Top

Recipe 6.5 Creating a Default Route in RIP

6.5.1 Problem

You want RIP to propagate a default route.

6.5.2 Solution

There are two ways to get RIP to propagate a default route. The preferred method is to use the *default-information originate* command as follows:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#ip route 0.0.0.0 0.0.0.0  172.25.1.1
Router1(config)#router rip
Router1(config-router)#default-information originate
Router1(config-router)#end
Router1#
```

In simple situations, you can accomplish the same thing by just redistributing a static route:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#ip route 0.0.0.0 0.0.0.0  172.25.1.1
Router1(config)#access-list 7 permit 0.0.0.0
Router1(config)#router rip
Router1(config-router)#redistribute static
Router1(config-router)#distribute-list 7 out static
Router1(config-router)#end
Router1#
```

6.5.3 Discussion

There are two main advantages to using *default originate* instead of simply redistributing static routes. The first is that you may have other static routes on your router that you do not want to distribute, or that you want to distribute with a different default metric. In this case, if you just use *redistribute static*, you will need to filter out the unwanted routes using route maps, as shown in Recipe 6.4.

The other important advantage is that the *default originate* option lets you create a *conditional default* route. This means that you can configure the router to create and distribute a default route only if some other route is present. Usually this other route points to a distant network. If the route is present, it

indicates that the router is able to see enough of the outside world to be a reliable default router:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#access-list 20 permit 192.168.55.0
Router1(config)#route-map DEFAULTROUTE permit 10
Router1(config-route-map)#match ip address 20
Router1(config-route-map)#exit
Router1(config)#router rip
Router1(config-router)#default-information originate route-map DEFAULTROUTE
Router1(config-router)#end
Router1#
```

In this example, if the distant network `192.168.55.0` is present in the routing table, RIP will generate and distribute a default route. Usually you would use this type of configuration on the RIP router that forms a gateway to another network.

You can see how RIP distributes the default route with the *show ip rip database* command:

```
Router1#show ip rip database 0.0.0.0 0.0.0.0
0.0.0.0/0    redistributed
    [1] via 0.0.0.0,
Router1#
```

6.5.4 See Also

Recipe 6.4

Top

Recipe 6.6 Disabling RIP on an Interface

6.6.1 Problem

You want to prevent an interface from participating in RIP.

6.6.2 Solution

You can prevent an interface from participating in RIP with the following set of commands:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#access-list 12 deny any
Router1(config)#router rip
Router1(config-router)#passive-interface FastEthernet0/1
Router1(config-router)#distribute-list 12 in FastEthernet0/1
Router1(config-router)#end
Router1#
```

6.6.3 Discussion

As we discussed in Recipe 6.1 , you enable RIP on an interface with a *network* command. But, because RIP expects any networks you specify this way to follow address class rules, it is quite easy to inadvertently enable RIP on an interface that you don't want to use this protocol.

There are two important reasons that might lead you to disable RIP on a particular interface. First, if you are already running another protocol on a particular interface, the additional RIP traffic could consume important bandwidth resources. Second, there may be devices on a particular network segment that you do not trust. In this case, you want to make sure that you don't let untrusted equipment distribute routing information into your network. This is particularly important because some Unix workstations run RIP by default, but the administrators rarely devote much attention to making sure that any local static routes have the correct metric values. It is possible for one misconfigured workstation to completely disrupt routing for an entire network.

This recipe does two things to disable RIP. First, the *passive-interface* command tells RIP not to send any routing information out through the specified interface. But this does not prevent the router from listening for incoming routes. So we have also applied an inbound *distribute-list* command to the interface to prevent RIP from learning any routes this way.

To demonstrate this, we have applied the *passive interface* command, but not the *distribute-list* command:

```
Router1#show ip route rip
R     192.168.30.0/24 [120/1] via 172.25.2.2, 00:00:09, Serial0/0.2
      172.25.0.0/16 is variably subnetted, 6 subnets, 3 masks
R        172.25.25.2/32 [120/1] via 172.25.2.2, 00:00:09, Serial0/0.2
R    192.168.20.0/24 [120/1] via 172.22.1.4, 00:00:08, FastEthernet0/1
Router1#
```

As you can see, the router is still learning routes through the `FastEthernet0/1` port. A debug trace proves that although the router doesn't send any routing information out through this interface, it does receive information this way:

```
Router1#debug ip rip
RIP protocol debugging is on
Aug 11 02:35:33.403: RIP: sending v1 flash update to 255.255.255.255 via
FastEthernet0/0.1
Aug 11 02:35:33.403: RIP: build flash update entries
Aug 11 02:35:33.403:     subnet 0.0.0.0 metric 16
Aug 11 02:35:33.403: RIP: sending v1 flash update to 255.255.255.255 via Serial0/0.2
Aug 11 02:35:33.403: RIP: build flash update entries
Aug 11 02:35:33.403:     subnet 0.0.0.0 metric 1
Aug 11 02:35:33.403:     network 172.21.0.0 metric 1
Aug 11 02:35:39.012: RIP: received v1 update from 172.22.1.4 on FastEthernet0/1
Aug 11 02:35:39.012:       192.168.20.0 in 1 hops
```

When we add the distribute-list command inbound on the `FastEthernet0/1` interface, the unwanted route disappears from the routing table:

```
Router1#show ip route rip
R     192.168.30.0/24 [120/1] via 172.25.2.2, 00:00:04, Serial0/0.2
      172.25.0.0/16 is variably subnetted, 5 subnets, 2 masks
R        172.25.25.2/32 [120/1] via 172.25.2.2, 00:00:04, Serial0/0.2
Router1#
```

You can see the effects of both commands in the output of the *show ip protocols* command:

```
Router1#show ip protocols
Routing Protocol is "rip"
  Sending updates every 30 seconds, next due in 13 seconds
  Invalid after 180 seconds, hold down 180, flushed after 240
Outgoing update filter list for all interfaces is not set
Incoming update filter list for all interfaces is not set
  FastEthernet0/1 filtered by 12 (per-user), default is 12
Redistributing: rip
Default version control: send version 1, receive any version
  Interface             Send  Recv  Triggered RIP  Key-chain
  FastEthernet0/0.1     1     1 2
  Serial0/0.2           1     1 2
 Automatic network summarization is in effect
Maximum path: 4
Routing for Networks:
  172.22.0.0
```

```
    172.25.0.0
  Passive Interface(s):
    FastEthernet0/1
  Routing Information Sources:
    Gateway         Distance      Last Update
    172.25.1.7           120      00:00:08
    172.25.2.2           120      00:00:00
  Distance: (default is 120)

Router1#
```

Note that you could apply an inbound access list to an interface to prevent the router from receiving RIP updates from other devices. To do this, simply apply a filter to block UDP port 520. However, as we discuss in Chapter 19, you cannot use access control lists to filter outbound router packets that originate on the router. In any case, that method forces the router to look at every incoming packet. This can cause serious CPU problems on fast links. It is far more efficient to just let the RIP process do the filtering, as we have shown in this recipe.

Similarly, it is more efficient to use the passive interface command to prevent the router from sending routes out through an interface, rather than using an outbound distribute list that merely blocks all routes. This is because, with the passive interface command, the router doesn't have to compare the routes it wants to send to an access list to decide whether to send them. Instead, it just knows not to send any routes.

6.6.4 See Also

Recipe 6.2

Top

Recipe 6.7 Unicast Updates for RIP

6.7.1 Problem

You want to exchange routing information with one device on a network, but not with any others.

6.7.2 Solution

You can configure RIP to send its updates to a neighboring router using unicast instead of broadcast or multicast packets. This is useful in two situations. First, on Non-Broadcast Multiple Access (NBMA) networks, you can't use the standard broadcast or multicast methods for distributing information because the media doesn't support it. And second, sometimes you need to exchange routing information with one or more specific devices on a segment, but you don't trust the rest to give you reliable information. This feature is rarely used, but it can be extremely valuable in these types of situations:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#router rip
Router1(config-router)#passive-interface FastEthernet0/1
Router1(config-router)#neighbor 172.22.1.4
Router1(config-router)#end
Router1#
```

6.7.3 Discussion

This recipe uses the *passive-interface* command discussed in Recipe 6.4to prevent the router from sending routing information to the interface in general. Note that it does not prevent the router from receiving routing information from other devices on the segment. We will discuss how to solve that problem in a moment.

A debug trace helps to show how the unicast update option works:

```
Router1#debug ip rip
RIP protocol debugging is on
Router1#
Aug 11 02:41:13.632: RIP: sending v1 update to 255.255.255.255 via FastEthernet0/0.1
(172.25.1.5)
Aug 11 02:41:13.636: RIP: sending v1 update to 255.255.255.255 via Serial0/0.2 (172.
25.2.1)
Aug 11 02:41:13.644: RIP: sending v1 update to 172.22.1.4  via FastEthernet0/1 (172.
22.1.3)
```

Here you can see that this router sends its updates to the general broadcast address (`255.255.255.255`

) for all of the other interfaces, but for `FastEthernet0/1` , the update goes directly to `172.22.1.4` . This example uses RIP Version 1. If it used Version 2, updates would be sent using the multicast address `224.0.0.9` , instead of the general segment broadcast address. However, the unicast option for Version 2 would work exactly the same as shown here.

The output of the *show ip protocols* command includes information about any unicast neighbors:

```
Router1#show ip protocols
Routing Protocol is "rip"
  Sending updates every 30 seconds, next due in 21 seconds
  Invalid after 180 seconds, hold down 180, flushed after 240
  Outgoing update filter list for all interfaces is not set
  Incoming update filter list for all interfaces is not set
  Redistributing: rip
  Neighbor(s):
    172.22.1.4
  Default version control: send version 1, receive any version
    Interface             Send  Recv  Triggered RIP  Key-chain
    FastEthernet0/0.1     1     1 2
    Serial0/0.2           1     1 2
  Automatic network summarization is in effect
  Maximum path: 4
  Routing for Networks:
    172.22.0.0
    172.25.0.0
  Passive Interface(s):
    FastEthernet0/1
  Routing Information Sources:
    Gateway         Distance      Last Update
    172.25.1.7           120      00:00:26
    172.25.2.2           120      00:00:14
    172.22.1.4           120      00:00:07
  Distance: (default is 120)

Router1#
```

As we noted in Recipe 6.6 , just making an interface passive does not prevent it from listening for updates. However, one of the most common reasons for using unicast neighbors with RIP is to ensure that the router accepts routing information only from specific devices on a segment. To do this, we need to configure the router to reject incoming RIP information from all other devices by applying an access list to the interface:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#access-list 101 permit udp host  172.22.1.4 any eq rip
Router1(config)#access-list 101 deny udp any any eq rip
Router1(config)#access-list 101 permit ip any any
Router1(config)#interface FastEthernet0/1
Router1(config-if)#ip access-group 101 in
Router1(config-if)#end
```

```
Router1#
```

6.7.4 See Also

Recipe 6.6

Top

Recipe 6.8 Applying Offsets to Routes

6.8.1 Problem

You want to modify the routing metrics for routes learned from or distributed into RIP.

6.8.2 Solution

You can modify the RIP metrics for a list of routes learned through a particular interface with the *offset-list* configuration command:

```
Router2#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router2(config)#access-list 22 permit 192.168.20.0
Router2(config)#router rip
Router2(config-router)#offset-list 22 in 5 Serial0.1
Router2(config-router)#end
Router2#
```

A similar command changes the metrics for a list of routes as they are sent out through a specified interface:

```
Router2#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router2(config)#access-list 33 permit  192.168.30.0
Router2(config)#router rip
Router2(config-router)#offset-list 33 out  10 Serial0.1
Router2(config-router)#end
Router2#
```

6.8.3 Discussion

The *offset-list* command is most useful when you need RIP to distinguish between the costs of different links. By default, RIP looks only at the number of hops to the destination. But sometimes the longer path is significantly faster. For example, you might have a primary link that uses a T1 to get to a router in another building, and an Ethernet segment to get from there to another router that connects to a server. It's obviously better to take this primary link rather than a backup 56Kbps circuit that happens to connect directly to the last-hop router.

Other routing protocols (such as OSPF) address this problem by assigning a cost to each link and

adding up the costs to find the best path. But RIP has only a metric. So, if you want to ensure that one path is preferred over another shorter one, you have to modify the metrics.

Cisco routers give considerable flexibility in changing metrics by adding an offset. This allows you to change each route independently as it is received or sent. You can even affect routes according to which interface the router receives them through. Be extremely careful because you can only increase a metric, never decrease it, and the maximum metric value is only 16. When the metric reaches a value of 16, RIP considers the network to be unreachable.

Sometimes you want to ensure that a particular path is never used to reach a certain destination. In this case, you can apply an offset of 16, and the router will have to find a different path. The router will also allow you to apply an offset of 0, but this has no effect.

The *show ip protocols* command lists all of the offsets that are configured, including information about the access lists, interfaces, and the sizes of the offsets that will be applied:

```
Router2#show ip protocols
Routing Protocol is "rip"
  Sending updates every 30 seconds, next due in 25 seconds
  Invalid after 180 seconds, hold down 0, flushed after 240
  Outgoing update filter list for all interfaces is not set
  Incoming update filter list for all interfaces is not set
  Outgoing routes in Serial0.1 will have 10 added to metric if on list 33
  Incoming routes in Serial0.1 will have 5 added to metric if on list 22
  Redistributing: rip
  Default version control: send version 1, receive any version
    Interface             Send  Recv  Triggered RIP  Key-chain
    Ethernet0             1     1 2
    Loopback0             1     1 2
    Serial0.1             1     1 2
  Automatic network summarization is in effect
  Maximum path: 4
  Routing for Networks:
    172.25.0.0
    192.168.30.0
  Routing Information Sources:
    Gateway         Distance      Last Update
    172.25.2.1           120      00:00:12
  Distance: (default is 120)
Router2#
```

A debug trace shows the incoming and outgoing routes. Note that the trace always shows the metric values after applying the offset for both inbound and outbound updates:

```
Router2#debug ip rip
RIP protocol debugging is on
Aug 10 23:24:36: RIP: sending v1 update to 255.255.255.255 via Serial0.1
Aug 10 23:24:36: RIP: build update entries
Aug 10 23:24:36:        subnet 172.25.25.2 metric 1
```

```
Aug 10 23:24:36:          network 192.168.30.0 metric 11
Aug 10 23:24:48: RIP: received v1 update from 172.25.2.1 on Serial0.1
Aug 10 23:24:48:      0.0.0.0 in 1 hops
Aug 10 23:24:48:      172.22.0.0 in 1 hops
Aug 10 23:24:48:      172.25.1.0 in 1 hops
Aug 10 23:24:48:      192.168.20.0 in 7 hops
Router2#
```

Top

Recipe 6.9 Adjusting Timers

6.9.1 Problem

You want to tune your routing protocol performance to decrease the amount of time that the network takes to converge after a topology change.

6.9.2 Solution

RIP has several timers that control how often it sends updates, how long it takes to remove a bad route, etc. You can adjust these values with the *timers basic* configuration command:

```
Router2#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router2(config)#router rip
Router2(config-router)#timers basic 20 80 80 120
Router2(config-router)#end
Router2#
```

6.9.3 Discussion

The *timers basic* command controls all of the adjustable timers for RIP:

```
Router2(config-router)#timers basic 20 80 80 120
```

The four arguments are, in order: the update period, the invalid route timer, the hold-down timer, and the flush timer. All of these times are in seconds.

The update period controls how often the router sends updates to its neighbors. The default update period is 30 seconds. Reducing this period can help to improve convergence times. However, you have to remember that RIP sends the entire routing table in each update cycle. If the routing table is large, reducing this period too much can cause serious bandwidth loading problems on slower links.

The invalid route timer controls how long the router will wait before declaring a particular route invalid. If the route disappears from the routing updates received from neighboring routers for this length of time, the router will mark it invalid. The router will set the RIP metric for invalid routes to 16, or unreachable, when distributing them. It is important to remember that the router doesn't remove invalid routes from its own routing table and will continue to distribute them to other routers.

The default invalid route timer value is 180 seconds. Most references advise making this value at least

three times the update period. Shorter invalid route timers can cause instability problems. But if you make it too much longer, the network won't respond well to topology changes.

The hold-down timer for a particular route starts when the router gets a routing update saying that the route is inaccessible. This could happen, for example, if another router's invalid route timer for this route has expired. It can also happen as a result of a triggered update indicating that a particular interface pointing to this network has gone down.

The router will keep the unreachable route in its table and distribute it to other routers with an unreachable metric. After the hold-down timer expires, the router will delete the unreachable route and start to accept other routing information for this route.

The default hold-down period is 180 seconds. It is usually a good idea to keep the hold-down time the same as the invalid route timer to help ensure network stability.

The final parameter is the flush timer. This controls how long the router will keep a route in its routing table before purging it. The default flush period is 240 seconds. It must be greater than the hold-down time, otherwise routes can be flushed before the hold-down timer expires, which makes the hold-down time meaningless.

We recommend using extreme caution when adjusting RIP timers. The timers on every router in a RIP network must be equal or you will see terrible instability problems. Note that the RIP timers affect the entire RIP process, meaning that you can't set the timer values separately for different neighbors or interfaces. It isn't possible to slow down the update timers over a slow WAN link and make them shorter over a faster LAN link.

In this example, we wanted to make RIP converge faster after topology changes, so we decreased all of the timers:

```
Router2(config-router)#timers basic 20 80 80 120
```

The net result is that we have reduced the time to flush a bad route to two minutes from the default value of four. But it is important to notice that we had to decrease all of the timers to achieve this result without compromising overall network stability.

As we mentioned earlier, routers will send the entire routing table on every update cycle, so making the timers too short can cause congestion problems on slower links. However, you can get away with shorter update periods if you use route summarization or filtering, as shown in Recipe 6.15. You can also decrease bandwidth consumption while improving convergence times by configuring the routers to only send updates when there are changes, as in Recipe 6.11.

Usually people are interested in reducing these timers to improve convergence times. We don't recommend increasing them from the default values, because it will make the network respond too slowly to topology changes. If you have a problem with older or slower routers that are unable to receive RIP updates as quickly as they are sent, a better solution is to adjust the interpacket delay, as

shown in Recipe 6.10.

You can view the values for all of the configured RIP timers with the *show ip protocols* command:

```
Router2#show ip protocols
Routing Protocol is "rip"
  Sending updates every 20 seconds, next due in 8 seconds
  Invalid after 80 seconds, hold down 80, flushed after 120
  Outgoing update filter list for all interfaces is not set
  Incoming update filter list for all interfaces is not set
    Redistributing: rip
  Default version control: send version 1, receive any version
    Interface             Send  Recv  Triggered RIP  Key-chain
    Ethernet0             1     1 2
    Serial0.1             1     1 2
  Automatic network summarization is in effect
  Maximum path: 4
  Routing for Networks:
    172.25.0.0
    192.168.30.0
  Routing Information Sources:
    Gateway         Distance      Last Update
    172.25.2.1           120      00:00:14
  Distance: (default is 120)
Router2#
```

6.9.4 See Also

Recipe 6.10; Recipe 6.11; Recipe 6.15

Top

Recipe 6.10 Configuring Interpacket Delay

6.10.1 Problem

You want to slow down the rate at which a router sends the packets in a single update to ensure that slower devices aren't overwhelmed, resulting in a loss of data.

6.10.2 Solution

Use the *output-delay* configuration command to adjust the interpacket delay of the RIP protocol:

```
Router2#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router2(config)#router rip
Router2(config-router)#output-delay 10
Router2(config-router)#end
Router2#
```

6.10.3 Discussion

If the routing table distributed in each update is small, it will fit into a single packet. But the maximum size for a RIP update packet is 512 bytes. Each packet has an 8-byte UDP header, and 4 bytes of general RIP information. The remaining 500 bytes of the packet carry the actual routes. Each individual route takes 20 bytes, so there can be at most 25 routes in one packet. If you have more than 25 routes in your routing table, RIP must send it as multiple packets.

The problem with this is that some devices can't process large amounts of incoming routing information quickly, and they wind up missing some of the routing information. This is particularly true for older legacy equipment and routers with slow processors or insufficient memory. If you find that one or more of the neighboring routers has a mysteriously incomplete routing table, it might help to increase the interpacket spacing. The other reason for increasing the output delay is to prevent bursts of RIP traffic from causing network congestion.

For example, in Frame Relay networks, if you burst above the Committed Information Rate (CIR), the extra packets may be marked as Discard Eligible (DE), which means that they might get dropped. This can cause serious problems, but spreading out this burst of RIP traffic might prevent them from occurring.

The *output-delay* command allows you to specify the interpacket delay in milliseconds. The default is 0,

meaning that the router will send packets as fast as it can. You can specify any delay value between 8 and 50 milliseconds with this command. It is not necessary to specify a corresponding delay on neighboring routers.

The output of the *show ip protocols* command includes the output interpacket delay if you configure any non-default value:

```
Router2#show ip protocols
Routing Protocol is "rip"
  Sending updates every 60 seconds, next due in 17 seconds
  Invalid after 180 seconds, hold down 180, flushed after 240
  Output delay 10 milliseconds between packets
  Outgoing update filter list for all interfaces is not set
  Incoming update filter list for all interfaces is not set
  Redistributing: rip
  Default version control: send version 1, receive any version
    Interface             Send  Recv  Triggered RIP  Key-chain
    Ethernet0             1     1 2
    Serial0.1             1     1 2
  Automatic network summarization is in effect
  Maximum path: 4
  Routing for Networks:
    172.25.0.0
    192.168.30.0
  Routing Information Sources:
    Gateway         Distance      Last Update
    172.25.2.1           120      00:00:23
  Distance: (default is 120)
Router2#
```

6.10.4 See Also

Chapter 10

Top

Recipe 6.11 Enabling Triggered Updates

6.11.1 Problem

You want to reduce RIP bandwidth requirements by configuring routers to forward only changes made to the routing table instead of forwarding the entire routing table.

6.11.2 Solution

The *ip rip triggered* interface configuration command tells the router to only send those parts of the RIP database that have changed, instead of the entire database on each RIP update cycle:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#interface Serial0/0.2
Router1(config-subif)#ip rip triggered
Router1(config-subif)#end
Router1#
```

Be sure to enable triggered updates on the adjacent router as well:

```
Router2#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router2(config)#interface Serial0.1
Router2(config-subif)#ip rip triggered
Router2(config-subif)#end
Router2#
```

> You must enable this feature on both of the routers sharing the point-to-point link, or you risk losing all routing information.

6.11.3 Discussion

This feature is based on RFC 2091, although it doesn't implement all of the features described in that RFC (in particular, RFC 2091 includes extensions to the Novell IPX RIP and SAP update process that Cisco does not implement). You can use it with both RIP Versions 1 and 2. It became available starting in IOS Version 12.0(1)T.

Triggered updates are available only for point-to-point serial interfaces. They are particularly useful for

slower serial WAN links where bandwidth is an issue. This feature is also used with dial-on-demand interfaces where it can save money by eliminating periodic RIP updates that would keep the link unnecessarily active.

When you enable this feature on an interface, the router will send routing updates only in a few well-defined situations. It will send a full copy of the database when it is first powered on to ensure that there is at least one update. The router will also send a full copy of the database if the neighboring router requests it. If the triggered interface itself goes up or down, the router will send a partial database to update the neighboring router on the changes since it last connected. And, in a relatively stable network, it will only send updates indicating incremental changes to the RIP database.

In a stable network, if there are no changes to the routing table, there are no updates. You can see the time since the last update with the *show ip protocols* command. In this case, from a very stable network, over 12 hours have elapsed since the last update:

```
Router2#show ip protocols
Routing Protocol is "rip"
  Sending updates every 60 seconds, next due in 52 seconds
  Invalid after 180 seconds, hold down 0, flushed after 240
Outgoing update filter list for all interfaces is not set
  Incoming update filter list for all interfaces is not set
  Redistributing: rip
  Default version control: send version 1, receive any version
    Interface             Send  Recv  Triggered RIP  Key-chain
    Ethernet0             1     1 2
    Serial0.1             1     1 2         Yes
  Automatic network summarization is in effect
  Maximum path: 4
  Routing for Networks:
    172.25.0.0
    192.168.30.0
  Routing Information Sources:
    Gateway         Distance      Last Update
    172.25.2.1           120      12:08:06
  Distance: (default is 120)
Router2#
```

If you combine this feature with route filtering and summarization (as discussed in Recipe 6.2 and Recipe 6.15), you can hide any instability that might exist elsewhere in the network. This is particularly useful for dialup links when you don't want to dial just because of a topology change elsewhere in the network.

The *show ip rip database* command marks routes learned via triggered interfaces as permanent:

```
Router2#show ip rip database 192.168.20.0 255.255.255.0
192.168.20.0/24
    [7] via 172.25.2.1, 00:02:43  (permanent), Serial0.1
   * Triggered Routes:
```

```
     - [7] via 172.25.2.1, Serial0.1
Router2#
```

The terminology is slightly confusing because this route is clearly not permanent in the same way as a static route. But it is not like normal RIP database entries, which are subject to the hold-down and flush timers. This is why the route is marked this way.

The following debug trace shows a triggered update for a route that became inaccessible. We ran this debug for 20 minutes, and this was the only event it recorded:

```
Aug 11 13:18:25: RIP: received v1 triggered update from 172.25.2.1 on Serial0.1
Aug 11 13:18:25: RIP: sending v1 ack to 172.25.2.1 via Serial0.1, seq# 15
Aug 11 13:18:25:      192.168.20.0 in 16 hops (inaccessible)
```

6.11.4 See Also

Recipe 6.2; Recipe 6.15

Top

Recipe 6.12 Increasing the RIP Input Queue

6.12.1 Problem

You want to increase the size of the RIP input queue to prevent your low-speed router from losing routing information.

6.12.2 Solution

To increase the size of the shared RIP queue, use the *input-queue* configuration command:

```
Router2#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router2(config)#router rip
Router2(config-router)#input-queue 200
Router2(config-router)#end
Router2#
```

6.12.3 Discussion

This command allows you to control how much incoming RIP update information the router can hold while it processes and integrates this information into its routing table. Sometimes a router simply can't keep up with all of the information that it receives. This is most likely to be the case for less powerful routers on a busy network with many routes.

Bear in mind that each RIP update packet can hold up to 25 routes, and the default queue size is more than adequate to hold this many routes. The input queue size is likely to be a problem only if you have many times this number of routes, or if you have many routers all sharing the same segment. If this is the case, and you find that a less powerful router is randomly losing routes from its table, it is relatively safe and easy to increase this queue depth.

The default value is 50. Here we have increased the queue depth to 200, which is a good starting point if you think that you have a queue depth problem. You can set this value to anything from 0 to 1024, although it is not clear why you would want to decrease the queue depth.

Recipe 6.10 shows another solution to this same problem. Instead of increasing the queue size on the slower router, you may opt to change the interpacket delay on the faster routers. And, in some cases, it may be necessary to combine both of these solutions.

6.12.4 See Also

Recipe 6.10

Top

Recipe 6.13 Configuring RIP Version 2

6.13.1 Problem

You want to use the more flexible features of RIP Version 2.

6.13.2 Solution

By default, Cisco routers will listen for both RIP Version 1 and 2 packets, but they will only send Version 1. If you want to configure the router to send and receive only Version 2 RIP packets, use the *version 2* configuration command:

```
Router2#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router2(config)#router rip
Router2(config-router)#version 2
Router2(config-router)#network 172.25.0.0
Router2(config-router)#network 192.168.30.0
Router2(config-router)#end
Router2#
```

You can also enable RIP Version 2 sending and receiving separately for individual interfaces:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#interface Serial0/0.2
Router1(config-subif)#ip rip send version 2
Router1(config-subif)#ip rip receive version 2
Router1(config-subif)#end
Router1#
```

6.13.3 Discussion

By default, the router will listen for Version 1 and 2 packets but it will only send Version 1. You can run both versions simultaneously on the same interface. However, the router will then have to transmit all updates using both versions, which can cause an unnecessary burden both for the router and the network. In general, RIP Version 2 is preferred, particularly because of the support for modern IP features such as variable-length subnet masks:

```
Router2#show ip protocols
Routing Protocol is "rip"
```

```
Sending updates every 60 seconds, next due in 19 seconds
Invalid after 180 seconds, hold down 0, flushed after 240
Output delay 10 milliseconds between packets
Outgoing update filter list for all interfaces is not set
Incoming update filter list for all interfaces is not set
  Redistributing: rip
Default version control: send version 2, receive version 2
  Interface             Send  Recv  Triggered RIP  Key-chain
  Ethernet0              2     2
  Serial0.1              2     2          Yes
Automatic network summarization is in effect
Maximum path: 4
Routing for Networks:
  172.25.0.0
  192.168.30.0
Routing Information Sources:
  Gateway         Distance      Last Update
  172.25.2.1           120      01:04:31
  Distance: (default is 120)
Router2#
```

This command shows that the router is sending and receiving only Version 2 on both interfaces. In this case, the router will reject any incoming RIP Version 1 packets on either of these ports. So you need to make sure that the neighboring routers on these interfaces are also running Version 2.

The second configuration example leaves the default configuration for all of the interfaces except `Serial0/0.2`, which sends and receives only RIP Version 2:

```
Router1#show ip protocols
Routing Protocol is "rip"
  Sending updates every 30 seconds, next due in 11 seconds
  Invalid after 180 seconds, hold down 0, flushed after 240
  Outgoing update filter list for all interfaces is not set
  Incoming update filter list for all interfaces is not set
  Redistributing: static, rip
  Default version control: send version 1, receive any version
    Interface             Send  Recv  Triggered RIP  Key-chain
    Serial0/0.2            2     2          Yes
    FastEthernet0/1        1     1 2
   Automatic network summarization is in effect
  Maximum path: 4
  Routing for Networks:
    172.22.0.0
    172.25.0.0
  Routing Information Sources:
    Gateway         Distance      Last Update
    172.25.2.2           120      14:40:28
    172.22.1.4           120      01:06:11
  Distance: (default is 120)
Router1#
```

Debug traces show which version the router sends and receives for every update:

```
Router2#debug ip rip
RIP protocol debugging is on
Router2#
Aug 11 14:38:54: RIP: received v2 update from 172.25.2.1 on Serial0.1
Aug 11 14:38:54:      0.0.0.0/0 via 0.0.0.0 in 1 hops
Aug 11 14:38:54:      172.22.0.0 /16 via 0.0.0.0 in 1 hops
Aug 11 14:38:54:      172.25.0.0/16 via 0.0.0.0 in 1 hops
Aug 11 14:38:54:      172.25.1.0/24 via 0.0.0.0 in 1 hops
Aug 11 14:38:54:       172.25.25.1/32 via 0.0.0.0 in 1 hops
Aug 11 14:38:57: RIP: sending v2 update to 224.0.0.9 via Serial0.1
Aug 11 14:38:57: RIP: build update entries
Aug 11 14:38:57:        172.25.25.2/32 via 0.0.0.0, metric 1,   tag 0
Aug 11 14:38:57:        192.168.30.0/24 via 0.0.0.0, metric 11, tag 0
Router2#
```

If you configure a router to listen only for one version, however, it will just drop any packets of the other version that it receives:

```
Router2#debug ip rip
RIP protocol debugging is on
Router2#
Aug 11 14:43:11: RIP: ignored v1 packet from 172.25.2.1 (illegal version)
Aug 11 14:43:11: RIP: received v2 update from 172.25.2.1 on Serial0.1
```

Top

Recipe 6.14 Enabling RIP Authentication

6.14.1 Problem

You want to authenticate your RIP traffic to ensure that unauthorized equipment cannot affect how traffic is routed through your network.

6.14.2 Solution

The following set of commands enables plain-text RIP authentication:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#key chain ORA
Router1(config-keychain)#key 1
Router1(config-keychain-key)#key-string oreilly
Router1(config-keychain-key)#exit
Router1(config-keychain)#exit
Router1(config)#interface FastEthernet0/0.1
Router1(config-subif)#ip rip authentication key-chain ORA
Router1(config-subif)#ip rip authentication mode text
Router1(config-subif)#end
Router1#
```

For greater security, Cisco routers can also use MD5-based authentication:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#key chain ORA
Router1(config-keychain)#key 1
Router1(config-keychain-key)#key-string oreilly
Router1(config-keychain-key)#exit
Router1(config-keychain)#exit
Router1(config)#interface FastEthernet0/0.1
Router1(config-subif)#ip rip authentication key-chain ORA
Router1(config-subif)#ip rip authentication mode md5
Router1(config-subif)#end
Router1#
```

6.14.3 Discussion

RIP authentication is one of the protocol enhancements that appeared in Version 2 of the protocol.

The first configuration example in this recipe uses plain-text authentication. In general, we recommend using the MD5 authentication because the plain-text version is far too easy to break. If you want to set up

authentication to ensure that you receive updates only from the appropriate devices, you should use the safer MD5 version. The only reason to consider the less secure plain-text version is if some of the RIP devices cannot support MD5. Because the RFC for RIP Version 2 only describes plain-text authentication, non-Cisco devices may not support MD5 authentication.

Both forms of RIP authentication help to ensure that only legitimate network equipment is allowed to take part in RIP updates. This is particularly important if you have network segments containing foreign devices that may corrupt the routing tables. This could happen because of malice, but it's also relatively easy for a misconfigured Unix workstation running the *routed* program to cause serious global routing problems.

When you enable plain-text authentication, the first route field in each update packet contains the authentication string instead of a route. This implies that each update packet can then hold a maximum of 24 route entries. Because the MD5 authentication scheme carries more information, it uses the first and last route fields in each update packet. So this leaves a maximum of 23 route entries per update packet.

In this example, the key is applied to an interface. This allows you to specify a different key for each network segment. However, there is nothing to stop you from using the same key on more than one interface, or even using a single key throughout the network.

The following debug traces were taken with authentication enabled. The first trace shows plain-text authentication and includes the password:

```
Router1#debug ip rip
RIP protocol debugging is on
Aug 12 02:08:03.386: RIP: received packet with text authentication oreilly
Aug 12 02:08:03.390: RIP: received v2 update from 172.25.1.7 on FastEthernet0/0.1
```

The second trace shows an update containing MD5 authentication. In this case, the router is not able to decode the authentication string. Instead, it compares the encrypted password string with the encrypted version of its own password to see if they match. There are no known methods to uniquely reverse MD5 encryption:

```
Router3#debug ip rip
RIP protocol debugging is on
Aug 11 22:14:50 EDT: RIP: received packet with MD5 authentication
Aug 11 22:14:50 EDT: RIP: received v2 update from 172.25.1.5 on Ethernet0
```

The *show ip protocols* command includes information about the authentication key chains:

```
Router3#show ip protocols
Routing Protocol is "rip"
  Sending updates every 30 seconds, next due in 16 seconds
  Invalid after 180 seconds, hold down 180, flushed after 240
  Outgoing update filter list for all interfaces is
  Incoming update filter list for all interfaces is
  Redistributing: rip
```

```
  Default version control: send version 2, receive version 2
    Interface         Send  Recv    Key-chain
    Ethernet0         2     2       ORA
  Routing for Networks:
    172.25.0.0
  Routing Information Sources:
    Gateway         Distance      Last Update
    172.25.1.5           120      00:00:01
  Distance: (default is 120)
Router3#
```

If the router receives a RIP update that has an incorrect key (or no key at all), it will discard the packet, as shown in the following debug trace:

```
Router3#debug ip rip
RIP protocol debugging is on
Aug 11 22:17:07 EDT: RIP: ignored v2 packet from 172.25.1.5 (invalid authentication)
```

We will discuss key management schemes such as setting key lifetimes and using multiple keys when we look at EIGRP authentication in Chapter 7. The key management systems are identical in both cases.

6.14.4 See Also

Chapter 7

Top

Recipe 6.15 RIP Route Summarization

6.15.1 Problem

You want to decrease the size of your routing tables to improve the stability and efficiency of the routing process.

6.15.2 Solution

You can manually configure address summarization on an individual interface with the *ip summary-address rip* configuration command:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#interface Serial0/0.2
Router1(config-subif)#ip summary-address rip 172.25.0.0 255.255.0.0
Router1(config-subif)#end
Router1#
```

By default, RIP Version 2 will summarize groups of subnets into classful network routes. You can disable this automatic summarization with the *no auto-summary* configuration command:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#router rip
Router1(config-router)#no auto-summary
Router1(config-router)#end
Router1#
```

6.15.3 Discussion

RIP Version 2 automatically summarizes along classful network boundaries. If your router sees that several subnets of the same network all use the same path, and that there are no subnets of this network using a different path, it automatically summarizes this information. The routes to the individual subnets are also suppressed. Any devices downstream from this router will see only a summary route (such as `172.25.0.0/16`) instead of all of the individual subnets (such as `172.25.1.0/24`, `172.25.2.16/28`, and so forth).

The downstream devices don't need to know that they are seeing summary routes instead of the actual individual routes, so summarizing works well even with legacy equipment.

However the *auto-summary* feature only works along classful network boundaries. For example, it will not summarize a group of networks such as `192.168.16.0/24`, through `192.168.31.0/24` into `192.168.16.0/20` because they are all separate Class C networks. RIP cannot advertise such supernets because they are not classful.

The interface-specific summarization command became available in IOS Version 12.0(6)T. Configuring summary address on an interface tells the router to send summary information out through a specific interface. All routers that are downstream from the configured interface will see only the summary route and none of the constituent subnet routes. As long as any one of these subnet routes are valid, the router will propagate the summary information. But, when the last subnet that is part of the summarized range disappears, the router will stop sending the summary route through the configured interface.

You cannot configure more than one summary address from the same classful network address on an interface. For example, the following is invalid because both of the summary addresses are subnets of `172.25.0.0/16`:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#interface Serial0/0.2
Router1(config-subif)#ip summary-address rip 172.25.16.0 255.255.240.0
Router1(config-subif)#ip summary-address rip 172.25.35.0 255.255.255.0
IP-RIP:No summary for 172.25.35.0/24.Summary 172.25.16.0/20 already on interface
Router1(config-subif)#end
Router1#
```

However, either of these summarizations would be fine on its own. You can configure several different summary addresses, as long as they are for different networks:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#interface Serial0/0.2
Router1(config-subif)#ip summary-address rip 172.25.16.0 255.255.240.0
Router1(config-subif)#ip summary-address rip 172.26.35.0 255.255.255.0
Router1(config-subif)#ip summary-address rip 10.101.0.0 255.255.0.0
Router1(config-subif)#end
Router1#
```

Route summarization has two important advantages. First, it reduces the size and complexity of the routing table, which in turn reduces the amount of memory and processing required on the downstream routers. Second, because all information about the individual subnets is suppressed, the downstream routers will not know or care about flapping interfaces in another part of the network. These two factors combine to improve overall network stability.

Suppose you have a routing table that looks like this before summarization:

```
Router2#show ip route rip
```

```
R     172.21.0.0/16 [120/1] via 172.25.2.1, 00:00:01, Serial0.1
R     172.22.0.0/16 [120/1] via 172.25.2.1, 00:00:01, Serial0.1
      172.25.0.0/16 is variably subnetted, 6 subnets, 3 masks
R       172.25.25.6/32 [120/2] via 172.25.2.1, 00:00:01, Serial0.1
R       172.25.25.1/32 [120/1] via 172.25.2.1, 00:00:01, Serial0.1
R       172.25.1.0/24 [120/1] via 172.25.2.1, 00:00:01, Serial0.1
R       172.25.0.0/16 [120/1] via 172.25.2.1, 00:00:01, Serial0.1
R*    0.0.0.0/0 [120/1] via 172.25.2.1, 00:00:02, Serial0.1
Router2#
```

If you enable automatic summarization, the router will replace all of the highlighted subnets with a single summary for the entire classful network address:

```
Router2#show ip route rip
R     172.21.0.0/16 [120/1] via 172.25.2.1, 00:00:01, Serial0.1
R     172.22.0.0/16 [120/1] via 172.25.2.1, 00:00:01, Serial0.1
      172.25.0.0/16 is variably subnetted, 3 subnets, 3 masks
R       172.25.0.0/16 [120/1] via 172.25.2.1, 00:00:01, Serial0.1
R*    0.0.0.0/0 [120/1] via 172.25.2.1, 00:00:01, Serial0.1
Router2#
```

You can see if a router is doing any summarization by looking at the *show ip protocols* command. This command shows both automatic summarization that is enabled by default, as well as any interface-specific manually configured summarization:

```
Router1#show ip protocols
Routing Protocol is "rip"
  Sending updates every 30 seconds, next due in 14 seconds
  Invalid after 180 seconds, hold down 0, flushed after 240
  Outgoing update filter list for all interfaces is not set
  Incoming update filter list for all interfaces is not set
  Redistributing: static, rip
  Default version control: send version 2, receive version 2
    Interface             Send  Recv  Triggered RIP  Key-chain
    FastEthernet0/0.1     2     2                    ORA
    Serial0/0.2           2     2
    FastEthernet0/1       2     2
   Automatic network summarization is in effect
  Address Summarization:
    172.25.0.0/16 for Serial0/0.2
  Maximum path: 4
  Routing for Networks:
    172.22.0.0
    172.25.0.0
  Routing Information Sources:
    Gateway         Distance      Last Update
    172.25.1.7           120      00:00:23
    172.25.2.2           120      00:00:00
    172.22.1.4           120      08:00:53
  Distance: (default is 120)
Router1#
```

The only thing you have to be careful of when using summarization is discontiguous networks. If you configure a router to advertise a summary route for downstream routers, ensure that no subnets in the summarized range exist in the downstream part of the network.

Top

Recipe 6.16 Route Tagging

6.16.1 Problem

You want RIP to include a tag when it distributes specific routes to prevent routing loops when redistributing between routing protocols.

6.16.2 Solution

RIP Version 2 allows you to tag external routes. For example, a static route configuration looks like this:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#ip route 0.0.0.0 0.0.0.0 172.25.1.1
Router1(config)#access-list 7 permit 0.0.0.0
Router1(config)#route-map TAGGING permit 10
Router1(config-route-map)#  match ip address 7
Router1(config-route-map)#  set tag 5
Router1(config-route-map)#  exit
Router1(config)#router rip
Router1(config-router)#redistribute static route-map TAGGING
Router1(config-router)#end
Router1#
```

6.16.3 Discussion

You can apply a route tag only to external routes: that is, routes that are not learned from RIP. The example shows a static route, but you can apply a tag to routes learned from other routing protocols in exactly the same way. For example, the following shows how to apply a tag to certain routes learned via EIGRP:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#access-list 7 permit 192.168.1.0
Router1(config)#route-map TAGGING permit 10
Router1(config-route-map)#  match ip address 7
Router1(config-route-map)#  set tag 5
Router1(config-route-map)#  set exit
Router1(config)#router eigrp 65530
Router1(config-router)#network 192.168.1.0
Router1(config-route-map)#  set exit
Router1(config)#router rip
```

```
Router1(config-router)#redistribute eigrp 65530 route-map TAGGING
Router1(config-router)#end
Router1#
```

RIP does not make use of the tags, it only distributes them. Once a route has a tag associated with it, every RIP Version 2 router will propagate this tag with the route. Tags are useful when redistributing routes into another routing process. For instance, if our RIPv2 network was being used as a transit network between two other networks, we could tag routes learned from one of these external networks and redistribute only those routes into the other network. This way, the RIP network would allow the two networks to talk to one another, but neither of the external networks could use the internal resources of the RIP network itself.

A debug trace shows the tagging in action:

```
Router1#debug ip rip
RIP protocol debugging is on
Aug 13 03:27:23.870: RIP: sending v2 update to 224.0.0.9 via Serial0/0.2
Aug 13 03:27:23.870: RIP: build update entries
Aug 13 03:27:23.870:     0.0.0.0/0 via 0.0.0.0, metric 1, tag 5
Aug 13 03:27:23.870:     172.22.0.0/16 via 0.0.0.0, metric 1, tag 0
Aug 13 03:27:23.874:     172.25.1.0/24 via 0.0.0.0, metric 1, tag 0
Aug 13 03:27:23.874:     172.25.25.1/32 via 0.0.0.0, metric 1, tag 0
Aug 13 03:27:23.874:     172.25.25.6/32 via 0.0.0.0, metric 2, tag 0
```

A similar trace on a downstream router would show that while the metric increments at each hop, the tag remains the same.

Here is an example that redistributes only the tagged routes into an attached EIGRP network. Note that you could easily construct a network where different tag values have different meanings. For example, you could make the tags equal to the routing process numbers of the various external networks. This would allow you to easily redistribute only the routes you want into other networks:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#router eigrp 11
Router1(config-router)#redistribute rip route-map TAGOUT
Router1(config-router)#exit
Router1(config)#route-map TAGOUT permit 10
Router1(config-route-map)#match tag 5
Router1(config-route-map)#route-map TAGOUT deny 20
Router1(config-route-map)#end
Router1#
```

RIPv2 supports 16-bit tags, which give a range of values between 0 and 65,535. As we discuss in Chapter 7 and Chapter 8, EIGRP and OSPF use 32-bit tags, for a range from 0 to 4,294,967,295. Of course, it is unlikely that you will need this many route tags.

6.16.4 See Also

Chapter 7; Chapter 8

Top

Chapter 7. EIGRP

Top

Introduction

Enhanced Interior Gateway Routing Protocol (EIGRP) is a Cisco proprietary routing protocol. You can only use it in an all-Cisco network, but EIGRP more than makes up for this deficiency by being easy to configure, fast, and reliable. A detailed discussion of the protocol's theory and operation is out of the scope of this book. If you are unfamiliar with EIGRP in general, or need more detail on how the protocol works, we recommend reading the relevant sections of IP Routing (O'Reilly).

Like RIP, EIGRP is based on a distance vector algorithm that determines the best path to a destination. But EIGRP uses a more complex metric than RIP's simple hop count. The EIGRP metric is based on the minimum bandwidth and net delay along each possible path, which means that EIGRP can accommodate larger networks than RIP. It also means that EIGRP needs a different algorithm for loop removal, because EIGRP can't simply increment the hop count to infinity to eliminate a loop, as RIP does. EIGRP uses a more sophisticated algorithm called Diffusing Update Algorithm (DUAL).

The DUAL algorithm ensures that every router can individually make sure that its routing table is always free from loops. EIGRP also allows the router to take advantage of several different possible paths, if they all have the same metric. This facilitates load sharing among equal cost links. Further, the EIGRP topology database on each router keeps track of higher cost candidates for the same destinations. This helps routing tables throughout the network to reconverge quickly after a topology change such as a link or router failure.

It is the sophisticated DUAL algorithm that distinguishes EIGRP from the earlier Cisco proprietary protocol, called Interior Gateway Routing Protocol (IGRP). IGRP is rarely used anymore, except for backward compatibility with older networks. Rather than implementing a new network with IGRP, we recommend using either EIGRP or OSPF. In fact, Cisco includes many useful features such as automatic two-way redistribution that make the migration from IGRP to EIGRP relatively straightforward.

EIGRP operates very efficiently over large networks. It achieves this efficiency in part by sending non-periodic updates. This means that, unlike RIP, EIGRP only distributes information about routes that have changed, and only when there is a change to report. The rest of the time, routers only exchange small "Hello" packets to verify that routing peers are still available. So, in a relatively stable network, EIGRP uses very little bandwidth. This is especially useful in WAN configurations.

It is also extremely efficient over LAN portions of a network. On each network segment, routers exchange routing information using multicast packets, which helps to limit bandwidth usage on segments that hold many routers. EIGRP uses multicast address `224.0.0.10`, sending packets as raw IP packets using protocol number 88. These multicast packets are always sent with a TTL value of 1 to

ensure that locally relevant routing information doesn't leak off the local segment and confuse routers elsewhere in the network.

Every router in an EIGRP network includes a topology table, which is a central feature of the DUAL algorithm. Every time a router receives a new piece of routing information from one of its neighbors, it updates the topology table. This helps to give it a reliable and up-to-date image of all of the connections in the network that are currently in use. Every destination subnet known to EIGRP appears in the topology table.

EIGRP includes many of the features such as Classless Inter-Domain Routing (CIDR) and Variable Length Subnet Masks (VLSM) that are needed in larger networks. But we suspect that this protocol owes most of its popularity to the fact that it is considerably easier to configure in medium-sized to large networks than other protocols with similar capabilities (such as OSPF).

Much of this chapter will discuss special features that Cisco has built into this protocol to help improve scalability. A detailed discussion of design guidelines for building scalable and reliable EIGRP networks is out of the scope of this book. Please refer to Designing Large-Scale LANs (O'Reilly) for information about efficient EIGRP architectures.

Top

Recipe 7.1 Configuring EIGRP

7.1.1 Problem

You want to run EIGRP on a simple network.

7.1.2 Solution

The following commands configure EIGRP on one router in a simple network:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#interface Ethernet0
Router1(config-if)#ip address 192.168.20.1 255.255.255.0
Router1(config-if)#exit
Router1(config)#interface Serial0.1 point-to-point
Router1(config-subif)#ip address 172.25.2.2 255.255.255.252
Router1(config-subif)#exit
Router1(config)#router eigrp 55
Router1(config-router)#network 172.25.0.0
Router1(config-router)#network 192.168.20.0
Router1(config-router)#end
Router1#
```

Naturally, you need to configure the other routers in this network to also exchange routing information using EIGRP and the same process number (55). For example:

```
Router2#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router2(config)#interface Serial0.1 point-to-point
Router2(config-subif)#ip address 172.25.2.1 255.255.255.252
Router2(config-subif)#exit
Router2(config)#router eigrp 55
Router2(config-router)#network 172.25.0.0
Router2(config-router)#end
Router2#
```

7.1.3 Discussion

This example shows how simple the basic EIGRP configuration is. To get the standard default functionality, you only need to enable EIGRP and add at least one network statement. In the example, we have set the EIGRP process ID numbers on both routers to 55:

```
Router1(config)#router eigrp 55
```

This process ID number, which is sometimes referred to as an Autonomous System Number (ASN), is just an arbitrary number between 1 and 65,535. The only restriction is that all of the routers that will be exchanging interior routing information via EIGRP must be configured with the same process number. You can configure multiple EIGRP instances on the same router by specifying different process ID numbers, but the router will keep them separate unless you configure redistribution in between the processes.

As we discuss in Chapter 9, BGP attaches much greater significance to an ASN, using it to label the networks that a path passes through. In BGP, the ASN must be globally unique. The EIGRP process ID number, on the other hand, has no significance outside of the AS.

The network statements in EIGRP serve a dual role, both defining which networks this router will distribute, and which interfaces will take part in the routing protocol. So, the *network 172.25.0.0* command in this example means that, if this router has any interfaces that are directly connected to subnets of `172.25.0.0`, then it will inject this information into the routing protocol. It also means that it will try to find EIGRP neighbor routers through these same interfaces.

It is important to remember that while EIGRP is a classless routing protocol, the argument of the network statement is classful by default. This isn't actually a problem, though, because you can separately prevent certain interfaces from taking part in the protocol, and you can define classless summarization of subnets along whatever boundaries you like. We will discuss these features in Recipe 7.6 and Recipe 7.7, respectively. There is also a classless version of the *network* command, which we will discuss later in this recipe.

The *show ip protocols* command allows you to look at the details of your EIGRP configuration:

```
Router1#show ip protocols
Routing Protocol is "eigrp 55"
  Outgoing update filter list for all interfaces is not set
  Incoming update filter list for all interfaces is not set
  Default networks flagged in outgoing updates
  Default networks accepted from incoming updates
  EIGRP metric weight K1=1, K2=0, K3=1, K4=0, K5=0
  EIGRP maximum hopcount 100
  EIGRP maximum metric variance 1
  Redistributing: eigrp 55
Automatic network summarization is in effect
Automatic address summarization:
   192.168.20.0/24 for Loopback0, Serial0.1
   172.25.0.0/16 for Ethernet0
     Summarizing with metric 128256
Maximum path: 4
Routing for Networks:
   172.25.0.0
   192.168.20.0
```

```
  Routing Information Sources:
    Gateway         Distance      Last Update
    172.25.2.1           90       00:01:49
  Distance: internal 90 external 170
Router2#
```

In this case, you can see that this router is using EIGRP process number 55 to redistribute routing information about `172.25.0.0` and `192.168.20.0`. It also shows several other useful pieces of information, such as what filters this router applies when sending and receiving routes, what external information is redistributed into EIGRP, and what neighboring devices we exchange information with. Most of the parameters shown in this particular output reflect the default values for EIGRP, but throughout this chapter you will find other examples showing several different useful variations.

One of the most useful EIGRP commands is *show ip eigrp neighbors*:

```
Router1#show ip eigrp neighbors
IP-EIGRP neighbors for process 55
H   Address                 Interface   Hold Uptime   SRTT   RTO  Q  Seq Type
                                        (sec)         (ms)       Cnt Num
0   172.25.2.1              Se0.1         13 00:00:01    1  2000  2  296
Router1#
```

By default, the router attempts to find adjacent routers on all interfaces included in your network statements. In this case, we see only one EIGRP neighbor router. The router will exchange routing information only with the active neighbors listed in this command.

The *show ip route eigrp* command lists the routes that have been learned through EIGRP:

```
Router1#show ip route eigrp
D     172.22.0.0/16 [90/2172416] via 172.25.2.1, 00:04:04, Serial0.1
      172.25.0.0/16 is variably subnetted, 6 subnets, 4 masks
D        172.25.25.6/32 [90/2300416] via 172.25.2.1, 00:04:04, Serial0.1
D       172.25.25.1/32 [90/2297856] via 172.25.2.1, 00:04:04, Serial0.1
D        172.25.1.0/24 [90/2172416] via 172.25.2.1, 00:04:04, Serial0.1
D       172.25.0.0/16 is a summary, 00:06:39, Null0
D     10.0.0.0/8 [90/4357120] via 172.25.2.1, 00:04:04, Serial0.1
Router1#
```

This output shows that we can reach the destination subnet `172.25.1.0/24` through the neighboring router at `172.25.2.1`, which is connected through interface `Serial0.1`. This route has an EIGRP metric value of `2172416` and an administrative distance of 90. Please refer to Chapter 5 for a more detailed discussion of administrative distance.

Starting in IOS Version 12.0(4)T, Cisco added a netmask argument to the *network* command, following a similar syntax to the corresponding OSPF command. This gives greater control over which interfaces will take part in the protocol, as well as what networks will be distributed into EIGRP:

```
Router1#configure terminal
```

```
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#router eigrp 55
Router1(config-router)#network 172.25.2.2 0.0.0.0
Router1(config-router)#network 192.168.20.0 0.0.0.255
Router1(config-router)#end
Router1#
```

Note that this command uses a wildcard rather than a netmask. So the first command specifies only the single address, `172.25.2.2/32`, while the second command includes anything that is a subnet of `192.168.20.0/24`.

The output of *show ip protocols* shows the change:

```
Router1#show ip protocols
Routing Protocol is "eigrp 55"
  Outgoing update filter list for all interfaces is not set
  Incoming update filter list for all interfaces is not set
    Serial0.1 filtered by (prefix-list) Inbound
  Default networks flagged in outgoing updates
  Default networks not accepted from incoming updates
  EIGRP metric weight K1=1, K2=0, K3=1, K4=0, K5=0
  EIGRP maximum hopcount 100
  EIGRP maximum metric variance 1
  Redistributing: static, eigrp 55
  Automatic network summarization is not in effect
  Maximum path: 4
  Routing for Networks:
    172.25.2.2/32
    192.168.20.0/24
  Routing Information Sources:
    Gateway         Distance      Last Update
    172.25.2.1              90      00:17:06
  Distance: internal 90 external 170
Router1#
```

This configuration can be slightly confusing because, for example, we have configured an EIGRP network statement for just the one address, `172.25.2.2/32`. Looking at the actual interface you can see that while its IP address does match the configured address, it belongs to a larger subnet, `172.25.2.0/30`. So, while we know that this will enable EIGRP for this interface, you might think that the router would advertise the host route, `172.25.2.2/32`, instead of the whole subnet, `172.25.2.0/30`. If you try it in practice, you will see that the router advertises the larger `/30` subnet. This is usually the desired behavior. However, if you want something else, Recipe 7.2 shows how to filter routes with EIGRP.

7.1.4 See Also

Recipe 7.2; Recipe 7.6; Recipe 7.7; Chapter 5

Top

Recipe 7.2 Filtering Routes with EIGRP

7.2.1 Problem

You want restrict which routes EIGRP propagates through the network.

7.2.2 Solution

You can filter the routes that EIGRP receives on a particular interface (or subinterface) using the *distribute-list in* command as follows:

```
Router2#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router2(config)#access-list 34 deny 192.168.30.0
Router2(config)#access-list 34 permit any
Router2(config)#router eigrp 55
Router2(config-router)#distribute-list 34 in Serial0.1
Router2(config-router)#end
Router2#
```

EIGRP also provides a *distribute-list out* command that allows you to filter the routes that are sent out through a particular interface (or subinterface):

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#access-list 57 permit 172.25.1.0
Router1(config)#access-list 57 deny any
Router1(config)#router eigrp 55
Router1(config-router)#distribute-list 57 out Serial0/0.2
Router1(config-router)#end
Router1#
```

7.2.3 Discussion

The best way to see the action of these *distribute-list* commands is to look at the routing tables both with and without the filters. In the example, this is how the routing table looked before we applied any distribute lists:

```
Router2#show ip route eigrp
D     192.168.30.0/24 [90/2300416] via 172.25.2.1, 00:00:06, Serial0.1
D     172.22.0.0/16 [90/2172416] via 172.25.2.1, 00:04:04, Serial0.1
      172.25.0.0/16 is variably subnetted, 6 subnets, 4 masks
```

```
D        172.25.25.6/32 [90/2300416] via 172.25.2.1, 00:04:04, Serial0.1
D        172.25.25.1/32 [90/2297856] via 172.25.2.1, 00:04:04, Serial0.1
D        172.25.1.0/24 [90/2172416] via 172.25.2.1, 00:04:04, Serial0.1
D       172.25.0.0/16 is a summary, 00:06:39, Null0
D     10.0.0.0/8 [90/4357120] via 172.25.2.1, 00:04:04, Serial0.1
Router2#
```

Then, after applying the inbound filter, you can see that network `192.168.30.0` is gone:

```
Router2#show ip route eigrp
D     172.22.0.0/16 [90/2172416] via 172.25.2.1, 00:00:08, Serial0.1
      172.25.0.0/16 is variably subnetted, 6 subnets, 4 masks
D        172.25.25.6/32 [90/2300416] via 172.25.2.1, 00:00:08, Serial0.1
D        172.25.25.1/32 [90/2297856] via 172.25.2.1, 00:00:08, Serial0.1
D        172.25.1.0/24 [90/2172416] via 172.25.2.1, 00:00:08, Serial0.1
D       172.25.0.0/16 is a summary, 00:08:42, Null0
D     10.0.0.0/8 [90/4357120] via 172.25.2.1, 00:00:08, Serial0.1
Router2#
```

You can use the *show ip protocols* command to see what filters have been applied to which interfaces, both inbound and outbound:

```
Router2#show ip protocols
Routing Protocol is "eigrp 55"
  Outgoing update filter list for all interfaces is not set
  Incoming update filter list for all interfaces is not set
    Serial0.1 filtered by 34 (per-user), default is 34
  Default networks flagged in outgoing updates
  Default networks accepted from incoming updates
  EIGRP metric weight K1=1, K2=0, K3=1, K4=0, K5=0
  EIGRP maximum hopcount 100
  EIGRP maximum metric variance 1
  Redistributing: eigrp 55
  Automatic network summarization is in effect
  Automatic address summarization:
    192.168.20.0/24 for Loopback0, Serial0.1
    172.25.0.0/16 for Ethernet0
      Summarizing with metric 128256
  Maximum path: 4
  Routing for Networks:
    172.25.0.0
    192.168.20.0
  Routing Information Sources:
    Gateway         Distance      Last Update
    172.25.2.1              90      00:02:10
  Distance: internal 90 external 170
Router2#
```

The second example in the solution section of this recipe shows an outbound distribute list. It is difficult to see the effect of an outbound filter from the router that has the filter, so we will apply this filter to the neighbor device.

Look back at the output of the previous *show ip route eigrp* command to see what the routing table looked like before applying this filter. Then, after applying the outbound *distribute-list* command on the neighboring router, the routing table looks like this:

```
Router2#show ip route eigrp
     172.25.0.0/16 is variably subnetted, 4 subnets, 4 masks
D       172.25.1.0/24 [90/2172416] via 172.25.2.1, 00:03:56, Serial0.1
Router2#
```

Note that we have applied an extremely restrictive outbound route filter. This technique is often used in WAN situations where there is only one path from the remote site to the rest of the network. In such cases, it is often possible to send only a few summary routes, or perhaps even a single default route, `0.0.0.0/0`.

Again, the *show ip protocols* command shows information about both the filters and the interfaces that they act on:

```
Router1#show ip protocols
Routing Protocol is "eigrp 55"
  Outgoing update filter list for all interfaces is not set
    Serial0/0.2 filtered by 57 (per-user), default is 57
  Incoming update filter list for all interfaces is not set
  Default networks flagged in outgoing updates
  Default networks accepted from incoming updates
  EIGRP metric weight K1=1, K2=0, K3=1, K4=0, K5=0
  EIGRP maximum hopcount 100
  EIGRP maximum metric variance 1
  Redistributing: eigrp 55
  Automatic network summarization is in effect
  Automatic address summarization:
    172.25.0.0/16 for FastEthernet0/1, Serial0/1
      Summarizing with metric 28160
    172.22.0.0/16 for FastEthernet0/0.1, Serial0/0.2, Loopback0
      Serial0/1
      Summarizing with metric 28160
    10.0.0.0/8 for FastEthernet0/0.1, Serial0/0.2, Loopback0
      FastEthernet0/1
      Summarizing with metric 3845120
  Maximum path: 4
  Routing for Networks:
    10.0.0.0
    172.22.0.0
    172.25.0.0
  Routing Information Sources:
    Gateway         Distance      Last Update
    10.1.1.1              90      00:04:45
    172.25.1.7            90      00:04:45
    172.25.2.2            90      00:04:45
    172.22.1.4            90      00:04:45
  Distance: internal 90 external 170
```

```
Router1#
```

You can also use prefix lists to filter routes with EIGRP. This technique is most commonly used for filtering routes with BGP. Prefix lists do essentially the same thing as the access lists that we have already discussed. But they give you a different way to approach filtering that is in some ways more in tune with how we think about routing. And, because of the highly granular control they offer, it is often much easier to configure a prefix list to do the same job as an access list. Further, in mixed BGP/EIGRP networks, it can be extremely convenient to be able to use the same method for both routing protocols:

```
Router2#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router2(config)#ip prefix-list Inbound seq 10 permit 10.0.0.0/8
Router2(config)#ip prefix-list Inbound seq 20 deny 10.0.0.0/8 ge 9
Router2(config)#ip prefix-list Inbound seq 30 permit 0.0.0.0/0 le 32
Router2(config)#router eigrp 55
Router2(config-router)#distribute-list prefix Inbound in Serial0.1
Router2(config-router)#end
Router2#
```

There are three lines in the prefix list called "Inbound" in this example. The first line permits the `10.0.0.0/8` network. The second line denies any network belonging to `10.0.0.0` that has a mask with 9 or more bits. The final line permits all other routes.

Again, to see how this works it is easiest to look at the routing table before and after applying the filter. So, in this case we will start with a routing table that looks like this:

```
Router2#show ip route eigrp
D     192.168.30.0/24 [90/2300416] via 172.25.2.1, 00:00:16, Serial0.1
      10.0.0.0/8 is variably subnetted, 3 subnets, 3 masks
D EX     10.0.0.0/8 [170/4357120] via 172.25.2.1, 00:00:16, Serial0.1
D        10.2.2.0/24 [90/2300416] via 172.25.2.1, 00:00:16, Serial0.1
D        10.1.1.0/30 [90/4357120] via 172.25.2.1, 00:00:16, Serial0.1
D*EX 0.0.0.0/0 [170/2172416] via 172.25.2.1, 00:00:16, Serial0.1
Router2#
```

After applying the filter, this routing table is reduced to the following:

```
Router2#show ip route eigrp
D     192.168.30.0/24 [90/2300416] via 172.25.2.1, 00:00:22, Serial0.1
D EX 10.0.0.0/8 [170/4357120] via 172.25.2.1, 00:00:22, Serial0.1
D*EX 0.0.0.0/0 [170/2172416] via 172.25.2.1, 00:00:22, Serial0.1
Router2#
```

7.2.4 See Also

Chapter 9

Top

Recipe 7.3 Redistributing Routes into EIGRP

7.3.1 Problem

You want to redistribute routes that were learned by other means into the EIGRP routing process.

7.3.2 Solution

The simplest way to redistribute routes into EIGRP uses the *redistribute* command:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#ip route 192.168.10.0 255.255.255.0 192.168.20.5
Router1(config)#router eigrp 55
Router1(config-router)#redistribute static
Router1(config-router)#end
Router1#
```

You can set the properties of the routes that are redistributed from another routing protocol with the *default-metric* command:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#router eigrp 55
Router1(config-router)#redistribute rip
Router1(config-router)#default-metric 1000 100 250 100 1500
Router1(config-router)#end
```

7.3.3 Discussion

The *show ip protocols* command tells you about any route redistribution that the protocol is performing:

```
Router1#show ip protocols
Routing Protocol is "eigrp 55"
  Outgoing update filter list for all interfaces is not set
  Incoming update filter list for all interfaces is not set
    Serial0.1 filtered by 34 (per-user), default is 34
  Default networks flagged in outgoing updates
  Default networks accepted from incoming updates
  EIGRP metric weight K1=1, K2=0, K3=1, K4=0, K5=0
  EIGRP maximum hopcount 100
```

```
    EIGRP maximum metric variance 1
    Redistributing: static, eigrp 55
    Automatic network summarization is in effect
    Automatic address summarization:
      192.168.20.0/24 for Loopback0, Serial0.1
      172.25.0.0/16 for Ethernet0
        Summarizing with metric 128256
    Maximum path: 4
    Routing for Networks:
      172.25.0.0
      192.168.20.0
    Routing Information Sources:
      Gateway         Distance      Last Update
      (this router)         90      00:05:00
      172.25.2.1            90      00:01:57
    Distance: internal 90 external 170
Router1#
```

If you look at the routing table of a downstream router, you can see that EIGRP has forwarded information about this static route:

```
Router2#show ip route eigrp
D     192.168.30.0/24 [90/156160] via 172.22.1.4, 00:00:02, FastEthernet0/1
D EX 192.168.10.0/24 [170/2195456] via 172.25.2.2, 00:00:01, Serial0/0.2
Router2#
```

There are two extremely important things to note in this output. The first is that the redistributed route is tagged as external, which is signified by the "EX" near the start of the line. An external route is any route that didn't originate with this routing protocol. This makes the information inherently less reliable than any internal route, so EIGRP also sets a higher administrative distance to ensure that internal EIGRP routes are always preferred over redistributed routes. This becomes extremely important when you have two or more redistribution points in your network that might be injecting the same routing information.

In this case, the administrative distance for the redistributed static route is 170, instead of the default EIGRP distance of 90. Recall fromChapter 5 that the default administrative distance for static routes is 1.

The second example in this recipe shows how to redistribute routes from a foreign routing protocol, instead of just static routes. The key difference is the *default-metric* command:

```
Router1(config-router)#redistribute rip
Router1(config-router)#default-metric 1000 100 250 100 1500
```

With static routes you don't need to configure a default metric, you can just use the *redistribute static* command. However, whenever you redistribute another routing protocol into EIGRP, you must configure the default metric. There is no default metric (strangely enough), so if you don't put it in, the router will not redistribute anything.

The parameters in the *default-metric* command allow EIGRP to construct an appropriate metric. Since none of this information is available from the foreign protocol, you have to specify it manually. The parameters are, in order:

Bandwidth

This value specifies the minimum bandwidth along the path in kilobits per second. It can have any value between 1 and 4,294,967,295.

Delay

This value defines the mean latency for the path in 10 microsecond units. It can be anything between 0 and 4,294,967,295.

Reliability

The reliability parameter is a numerical estimate of how likely the route and the path are to be available at any given moment. You can specify any value between 0 and 255, where 255 represents perfect 100% reliability.

Effective bandwidth (Loading)

This value is intended to provide a way of shifting traffic off of heavily loaded network links. You can give it a value between 0 and 255, where 255 represents 100% utilization.

Maximum Transmission Unit (MTU)

You can use this value to specify a path MTU to reach the foreign routing protocol. The range of values for this metric is between 0 and 4,294,967,295.

As it turns out, however, EIGRP doesn't use most of this information by default. If you look at the output of any *show ip protocols* command in this chapter, you will see a line that specifies the EIGRP metric weights:

```
EIGRP metric weight K1=1, K2=0, K3=1, K4=0, K5=0
```

EIGRP uses these K values as coefficients in an involved equation that specifies how to combine all of these different individual metrics into a single numerical value, the composite EIGRP metric. Note that only K1 and K3 are non-zero. The result is that, by default, EIGRP uses only bandwidth and delay when computing its metric. You can generally fill in just about anything for the other parameters in the *default-metric* command, and it won't make any difference.

While you can change these different K values using the *metric weights* command, we strongly advise against changing the defaults. These values were of some use in IGRP, and when Cisco introduced EIGRP (with its superior DUAL algorithm), they carried the parameters forward. However, it was discovered that in practice it was relatively easy to make routing extremely unstable by changing them. And it's almost impossible to make things any better by changing these weight values in EIGRP.

The *default-metric* command sets the metric values for all external routing protocols. If you need to specify different metrics for different protocols, you can put the same information on the *redistribute* command line as follows:

```
Router1(config-router)#redistribute rip metric 1000 100 250 100 1500
Router1(config-router)#redistribute ospf 99 metric  1500 10 255 10 1500
```

Table 7-1 shows all of the different protocols that you can redistribute into EIGRP using this method. You must specify each redistributed protocol separately.

Table 7-1. Valid redistribution protocols for EIGRP

keyword	Description
bgp	Border Gateway Protocol
connected	Connect interfaces
egp	Exterior Gateway Protocol
eigrp	Enhanced IGRP
igrp	Interior Gateway Routing Protocol
isis	ISO IS-IS Routing Protocol
mobile	Mobile routes
odr	On Demand stub routes
ospf	Open Shortest Path First
rip	Routing Information Protocol
static	Static routes

Actually, there is an important exception in this list. If a router has EIGRP and IGRP, both sharing the same process ID number, it will automatically redistribute between them. This is a convenient feature because it makes it relatively easy to migrate an IGRP network to EIGRP. However, if the EIGRP process number is not the same as the one used for IGRP, you need to configure the redistribution.

Sometimes you don't want to redistribute all of the routes from a particular external routing protocol, just some of them. In this case, you can apply a distribute list (as shown in Recipe 7.2) to ensure that only those routes that are redistributed from a particular protocol. For example, you might have several static routes on your router, but you only want to redistribute some of them. In that case, you can apply a distribute list to only the static routes:

```
Router1(config)#router eigrp 55
Router1(config-router)#redistribute static
Router1(config-router)#distribute-list 7 out static
```

This will apply access list number 7 to all of the static routes before distributing them. You can also use this technique when redistributing routes learned from other routing protocols. For example, you might want to filter the routes learned from OSPF before redistributing them into EIGRP. You can do this as follows:

```
Router1(config)#router eigrp 55
Router1(config-router)#redistribute ospf 99
Router1(config-router)#distribute-list 7 out ospf 99
```

7.3.4 See Also

Recipe 7.2; Chapter 5

Top

Recipe 7.4 Redistributing Routes into EIGRP Using Route Maps

7.4.1 Problem

You require greater control over the routes that are redistributed and their associated metrics and route tags.

7.4.2 Solution

You can use route maps to do more sophisticated redistribution of routes into EIGRP:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#ip route 192.168.10.0 255.255.255.0 172.22.1.4
Router1(config)#ip route 192.168.11.0 255.255.255.0 172.22.1.4
Router1(config)#ip route 192.168.12.0 255.255.255.0 172.22.1.4
Router1(config)#access-list 20 permit 192.168.10.0
Router1(config)#access-list 21 permit 192.168.11.0
Router1(config)#route-map STATIC permit 10
Router1(config-route-map)#  match ip address 20
Router1(config-route-map)#  set metric 56 100 255 1 1500
Router1(config-route-map)#  set tag 2
Router1(config-route-map)#exit
Router1(config)#route-map STATIC permit 20
Router1(config-route-map)#  match ip address 21
Router1(config-route-map)#  set metric 128 200 255 1 1500
Router1(config-route-map)#exit
Router1(config)#route-map STATIC deny 30
Router1(config-route-map)#exit
Router1(config)#router eigrp 55
Router1(config-router)#redistribute static route-map STATIC
Router1(config-router)#end
Router1#
```

7.4.3 Discussion

This recipe is extremely similar to Recipe 6.4 in the RIP chapter of this book. And, just as in that example, we use route maps to set not only metrics but also route tags for redistributed static routes. Please refer to Recipe 7.1 for a detailed discussion of how route maps work.

The one thing that you need to be careful of with EIGRP is that, as we discussed in Recipe 7.3, there is

no default default metric. So if you don't define EIGRP metrics for foreign routing protocols, EIGRP will not redistribute anything. This is not necessary for the static routes shown in the example, though.

7.4.4 See Also

Recipe 7.3; Recipe 6.4

Top

Recipe 7.5 Creating a Default Route in EIGRP

7.5.1 Problem

You want to propagate a default route within EIGRP.

7.5.2 Solution

You can configure EIGRP to propagate a default route by simply redistributing a static route to `0.0.0.0/0`, as follows:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#ip route 0.0.0.0 0.0.0.0  172.25.1.1
Router1(config)#access-list 7 permit 0.0.0.0
Router1(config)#router eigrp 55
Router1(config-router)#redistribute static
Router1(config-router)#distribute-list 7 out static
Router1(config-router)#end
Router1#
```

7.5.3 Discussion

This example actually shows two things. We have redistributed a simple static default route into EIGRP, as shown in Recipe 7.3. And we have also implemented an outbound route filter that only affects the static routes, as discussed in Recipe 7.2. Note that because of the *static* keyword on the *distribute-list* command, this distribute list applies only to static routes. So, if there are many static routes on this router, this feature ensures that we will only redistribute the default route.

If we go to a downstream router, you can see that EIGRP is forwarding this route, and that it is accepted as a candidate default route:

```
Router2#show ip route 0.0.0.0
Routing entry for 0.0.0.0/0, supernet
  Known via "eigrp 55", distance 170, metric 2172416, candidate default path, type
external
  Redistributing via eigrp 55
  Last update from 172.25.2.1 on Serial0.1, 00:02:16 ago
  Routing Descriptor Blocks:
  * 172.25.2.1, from 172.25.2.1, 00:02:16 ago, via Serial0.1
      Route metric is 2172416, traffic share count is 1
```

```
       Total delay is 20100 microseconds, minimum bandwidth is 1544 Kbit
       Reliability 255/255, minimum MTU 1500 bytes
      Loading 1/255, Hops 1
Router2#
```

You can look at the topology table to see how EIGRP classifies the default route:

```
Router2#show ip eigrp topology 0.0.0.0
IP-EIGRP (AS 55): Topology entry for 0.0.0.0/0
  State is Passive, Query origin flag is 1, 1 Successor(s), FD is 2172416
  Routing Descriptor Blocks:
  172.25.2.1 (Serial0.1), from 172.25.2.1, Send flag is 0x0
      Composite metric is (2172416/28160),  Route is External
      Vector metric:
        Minimum bandwidth is 1544 Kbit
        Total delay is 20100 microseconds
        Reliability is 255/255
        Load is 1/255
        Minimum MTU is 1500
        Hop count is 1
      External data:
        Originating router is 172.25.25.1
        AS number of route is 0
        External protocol is Static, external metric is 0
        Administrator tag is 0 (0x00000000)
        Exterior flag is set
Router2#
```

In this case, the command was issued on a downstream router that has received this default route via EIGRP. So it is shown as an external route. You can also see that the router `172.25.25.1` was responsible for introducing this external route into EIGRP, and that the external routing protocol is "static."

Recipe 7.10 shows another way of distributing a default gateway that doesn't mark the route as external.

7.5.4 See Also

Recipe 7.2; Recipe 7.3; Recipe 7.10

Top

Recipe 7.6 Disabling EIGRP on an Interface

7.6.1 Problem

You want to disable an interface from participating in EIGRP.

7.6.2 Solution

You can prevent an interface from participating in EIGRP by simply designating it as passive:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#router eigrp 55
Router1(config-router)#passive-interface Serial0/1
Router1(config-router)#end
Router1#
```

7.6.3 Discussion

The *passive-interface* command in EIGRP prevents directly connected routers from establishing an EIGRP neighbor relationship. Since they can't become neighbors, they will never exchange routing information. This is critically different from the way RIP behaves, as we saw in Chapter 6. In RIP, making an interface passive means that it will still accept routes, it just won't send them. But with EIGRP, a passive interface will not send or receive any routing information.

Furthermore, configuring one router to be passive means that it can't form an EIGRP adjacency relationship with any other routers through this interface. If there are only two routers on a link, you can disable EIGRP on that link by simply configuring one of the routers with a passive interface.

You can see the neighbor relationships with the following command:

```
Router1#show ip eigrp neighbors
IP-EIGRP neighbors for process 55
H   Address                 Interface   Hold Uptime   SRTT   RTO  Q  Seq Type
                                        (sec)         (ms)       Cnt Num
0   172.25.2.2              Se0/0.2       11 00:07:03 1563  5000  0  81
3   172.25.1.7              Fa0/0.1       77 00:18:17   11   200  0  348
2   172.22.1.4              Fa0/1         12 00:18:42    4   200  0  197
1   10.1.1.1                Se0/1          14 00:18:43    7   200  0  196
Router1#
```

If we then implement the *passive-interface* command on this router, as shown above, you can see that the neighbor disappears from the table:

```
Router1#show ip eigrp neighbors
IP-EIGRP neighbors for process 55
H   Address                 Interface   Hold Uptime   SRTT   RTO  Q  Seq Type
                                        (sec)         (ms)       Cnt Num
0   172.25.2.2              Se0/0.2       14 00:08:56 1563  5000  0  81
3   172.25.1.7              Fa0/0.1       69 00:20:10   11   200  0  348
2   172.22.1.4              Fa0/1         12 00:20:35    4   200  0  197
Router1#
```

The *show ip protocols* command lists all of the passive interfaces that are configured on this router:

```
Router1#show ip protocols
Routing Protocol is "eigrp 55"
  Outgoing update filter list for all interfaces is not set
    Redistributed static filtered by 7
  Incoming update filter list for all interfaces is not set
  Default networks flagged in outgoing updates
  Default networks accepted from incoming updates
  EIGRP metric weight K1=1, K2=0, K3=1, K4=0, K5=0
  EIGRP maximum hopcount 100
  EIGRP maximum metric variance 1
  Redistributing: static, eigrp 55
  Automatic network summarization is in effect
  Automatic address summarization:
    172.25.0.0/16 for FastEthernet0/1
      Summarizing with metric 28160
    172.22.0.0/16 for FastEthernet0/0.1, Serial0/0.2, Loopback0
      Summarizing with metric 28160
    10.0.0.0/8 for FastEthernet0/0.1, Serial0/0.2, Loopback0
      FastEthernet0/1
      Summarizing with metric 3845120
  Maximum path: 4
  Routing for Networks:
    10.0.0.0
    172.22.0.0
    172.25.0.0
  Passive Interface(s):
    Serial0/1
  Routing Information Sources:
    Gateway         Distance      Last Update
    172.25.1.7              90      00:09:57
    172.25.2.2              90      00:09:57
    172.22.1.4              90      00:09:57
  Distance: internal 90 external 170
Router1#
```

7.6.4 See Also

Chapter 6

Top

Recipe 7.7 EIGRP Route Summarization

7.7.1 Problem

You want to reduce the size of your routing tables to improve the stability and efficiency of the routing process.

7.7.2 Solution

The ip summary-address eigrp configuration command allows you to configure manual summary addresses on a per-interface basis:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#interface Serial0/0.2
Router1(config-subif)#ip summary-address eigrp 55 172.25.0.0 255.255.0.0
Router1(config-subif)#end
Router1#
```

By default, EIGRP will automatically summarize subnet routes into network-level routes. You can disable this with the no auto-summary configuration command:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#router eigrp 55
Router1(config-router)#no auto-summary
Router1(config-router)#end
Router1#
```

7.7.3 Discussion

Summarization is one of the most powerful features of EIGRP, and one of the most frequently overlooked ways to improve network efficiency. Unlike RIP, which summarizes along classful network boundaries, EIGRP uses CIDR, allowing you to summarize at any bit in the address as well as allowing supernets. And, while OSPF also allows this sort of summarization, as we will discuss in Chapter 8, OSPF can only summarize at the ABR. Conversely, EIGRP allows you to summarize at any router in the network. This means that you can have multiple hierarchical levels of address summarization with EIGRP, which can greatly improve the maximum size and efficiency of a large network that is designed properly to allow it.

You can see all of the summarization information, including which interfaces will send out summary addresses, using the show ip protocols command:

```
Router1#show ip protocols
Routing Protocol is "eigrp 55"
  Outgoing update filter list for all interfaces is not set
    Redistributed static filtered by 7
  Incoming update filter list for all interfaces is not set
  Default networks flagged in outgoing updates
  Default networks accepted from incoming updates
  EIGRP metric weight K1=1, K2=0, K3=1, K4=0, K5=0
  EIGRP maximum hopcount 100
  EIGRP maximum metric variance 1
  Redistributing: static, eigrp 55
  Automatic network summarization is not in effect
  Address Summarization:
    172.25.0.0/16 for Serial0/0.2
      Summarizing with metric 28160
  Maximum path: 4
  Routing for Networks:
    10.0.0.0
    172.22.0.0
    172.25.0.0
  Routing Information Sources:
    Gateway         Distance      Last Update
    10.1.1.1              90      1d23h
    172.25.1.7            90      00:00:57
    172.25.2.2            90      00:00:57
    172.22.1.4            90      00:00:57
  Distance: internal 90 external 170
Router1#
```

Note that when you summarize like this, the router doing the summarization will install a special route pointing to the null interface:

```
Router1#show ip route 172.25.0.0
<lines deleted for brevity>
D       172.25.0.0/16 is a summary, 00:00:23, Null0
Router1#
```

In this example, we have only summarized `172.25.0.0/16` on interface `Serial0/0.2`. However, it is important to remember that you can summarize several networks at the same time on a single interface by simply configuring all of the different summary addresses, as follows:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#interface Serial0/0.2
Router1(config-subif)#ip summary-address eigrp 55 172.25.0.0 255.255.0.0
Router1(config-subif)#ip summary-address eigrp 55 10.0.0.0 255.0.0.0 80
Router1(config-subif)#end
Router1#
```

When it summarizes addresses, EIGRP will automatically suppress all of the routes that are included in the summary. Of course, if there are no routes to summarize, the router won't distribute the summary address.

The metric of this summary route will be equal to the best metric of the routes being summarized. It is important to remember this because, if the route with the best metric goes away for any reason, EIGRP will change the metric of the summary. So, if the route with the best metric is unstable, it will make the summary route unstable. If you want to ensure that this doesn't happen, you can configure a static route within the summarized range and point it to a null interface. Then you must configure the router to redistribute this static route into EIGRP.

The following example shows a CIDR supernet summarization:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#interface Serial0/0.2
Router1(config-subif)#ip summary-address eigrp 55 0.0.0.0 0.0.0.0
Router1(config-subif)#end
Router1#
```

In this case, if there are any routes to distribute at all, EIGRP will distribute only the default route `0.0.0.0/0`, and suppress all of the individual routes. This is actually an extremely useful technique on low-speed WAN links, particularly when this link represents the only path to the rest of the network. In such cases, the remote site only needs to know that it can get to everything it needs through this link. Further, because routing is always done by taking the longest match first, if the remote site happens to have more specific routing information for a particular destination, it won't use this summary route.

You could accomplish the same thing by injecting a default route (as shown in Recipe 7.5) and filtering out everything except `0.0.0.0/0` using a distribute list (as in Recipe 7.2). But this summary address technique does both of these actions in a single step. Furthermore, with this technique, the default route appears in the routing table as an internal route:

```
Router2#show ip route eigrp
D*    0.0.0.0/0 [90/2172416] via 172.25.2.1, 00:00:30, Serial0.1
Router2#
```

7.7.4 See Also

Recipe 7.2; Recipe 7.5; Chapter 8

Top

Recipe 7.8 Adjusting EIGRP Metrics

7.8.1 Problem

You want to modify the routing metrics for routes learned via EIGRP.

7.8.2 Solution

You can use the *offset-list* configuration command to modify the metrics of routes that EIGRP learns through a particular interface:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#access-list 22 permit 192.168.30.0
Router1(config)#router eigrp 55
Router1(config-router)#offset-list 22 in 10000 Serial0.1
Router1(config-router)#end
Router1#
```

This command can also modify the EIGRP metrics of routes as the router sends them out through an interface:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#access-list 33 permit 192.168.30.0
Router1(config)#router eigrp 55
Router1(config-router)#offset-list 33 out 10000 Serial0.1
Router1(config-router)#end
Router1#
```

7.8.3 Discussion

This command simply adds a constant value to the metrics of all of the routes that are either sent or received through a particular interface. There are actually two other ways to modify metrics in EIGRP. Recall that the EIGRP metric is a combination of the aggregate delay and the minimum bandwidth along a path. So, instead of adding an offset to the entire metric, you can modify the bandwidth and delay separately as follows:

```
Router1(config)#interface Serial0.1
Router1(config-if)#bandwidth 56
Router1(config-if)#delay 1000
```

The *bandwidth* command takes an argument in kilobits per second, and will accept a value between 1 and 10,000,000Kbps. The *delay* command is measured in tens of microseconds, and can be anywhere between 1 and 16,777,215. In this case, we have specified a value of 1000, meaning a delay of 10,000 microseconds (10 milliseconds). You can see the current values for both of these parameters with the *show interface* command:

```
Router1#show interfaces serial0.1
Serial0.1 is up, line protocol is up
  Hardware is HD64570
  Internet address is 172.25.2.2/30
  MTU 1500 bytes, BW 1544 Kbit, DLY 20000 usec,
     reliability 255/255, txload 1/255, rxload 1/255
  Encapsulation FRAME-RELAY
Router1#
```

In this example, subinterface `Serial0.1` has the default values for a serial interface, a bandwidth of 1544Kbps (a T1), and a delay of 20,000 microseconds (20 milliseconds). It is always a good idea to check the current values before adjusting either the bandwidth or delay parameters, if only to make sure that you are moving them in the right direction.

We offer one important caution on adjusting the bandwidth parameter in particular. This same value also appears in the SNMP variable *ifSpeed* for this interface. This is often used by performance management software to define the total available bandwidth for the interface. Changing this number to fix an EIGRP issue might cause a problem for your performance management system.

One of the problems with adjusting the delay and bandwidth on the interface is that you can't use this to separately adjust inbound and outbound routing metrics. If you need this level of control, the offset list method discussed previously is the best way to achieve it.

You can see effect of an offset list in the output of the *show ip protocols* command:

```
Router1#show ip protocols
Routing Protocol is "eigrp 55"
  Outgoing update filter list for all interfaces is not set
  Incoming update filter list for all interfaces is not set
    Serial0.1 filtered by (prefix-list) Inbound
  Incoming routes in Serial0.1 will have 10000 added to metric if on list 22
  Default networks flagged in outgoing updates
  Default networks not accepted from incoming updates
  EIGRP metric weight K1=1, K2=0, K3=1, K4=0, K5=0
  EIGRP maximum hopcount 100
  EIGRP maximum metric variance 1
  Redistributing: static, eigrp 55
  Automatic network summarization is in effect
  Automatic address summarization:
    192.168.20.0/24 for Loopback0, Serial0.1
    172.25.0.0/16 for Ethernet0
      Summarizing with metric 128256
```

```
  Maximum path: 4
  Routing for Networks:
    172.25.0.0
    192.168.20.0
  Routing Information Sources:
    Gateway         Distance      Last Update
    172.25.2.1           90       00:02:09
  Distance: internal 90 external 170
Router1#
```

You can also see the difference it makes by looking at the routing tables. In this case, the route looked like this before we applied the offset list:

```
Router1#show ip route eigrp
D    192.168.30.0/24 [90/200416] via 172.25.2.1, 00:00:24, Serial0.1
```

As you can see, the metric has increased by 10,000 after applying the offset:

```
Router1#show ip route eigrp
D    192.168.30.0/24 [90/210416] via 172.25.2.1, 00:00:24, Serial0.1
```

Top

Recipe 7.9 Adjusting Timers

7.9.1 Problem

You wish to tune your EIGRP timers to improve network convergence.

7.9.2 Solution

There are two important EIGRP timers, the hello interval and the hold time. You can adjust both of these timers separately on each interface on a router as follows:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#interface Serial0.1
Router1(config-subif)#ip hello-interval eigrp 55 3
Router1(config-subif)#ip hold-time eigrp 55 9
Router1(config-subif)#end
Router1#
```

7.9.3 Discussion

One of the unique features of EIGRP is that you can adjust its timers separately on each interface. As we mentioned in Chapter 6, RIP requires you to adjust the timers identically on every interface of every device participating in RIP. In Chapter 8 you will see that while OSPF allows you to adjust the timers separately on each link, you have to make sure that it is the same on all routers on this link. But with EIGRP, you can adjust the timers on one router on a link independently of what you have configured on other interfaces on this router, or on other routers on this link.

EIGRP handles this by simply telling the other routers on the link what its parameters are. Therefore, if one router has a particular hello time of, say, 5 seconds, then all of the other routers on this link will expect to see a hello packet from this router every 5 seconds. This is true regardless of what the other routers have for their own parameters. The result is that when you adjust the timers on an interface on one router, you affect what its neighbors expect to see from it.

The default timer values for most interface types are 5 seconds for hellos and a 15-second hold timer. This means that the router will send out a hello packet to verify its neighbor relationships every 5 seconds. And, if it doesn't hear from a neighbor device, it will wait 15 seconds before declaring that neighbor down.

On multipoint interfaces with sub-T1 speeds, the default hello time is 60 seconds, with a hold time of 180 seconds. Note that the defaults always have a hold time equal to three times the hello time. It is a good rule of thumb to keep this three to one ratio if you choose to adjust your timers.

You can cause serious network stability problems if you don't adjust the hold and hello times together. In particular, if the hold time is less than the hello time, you will see frequent loss of neighbor status, causing instability. And if the hold time is too long, you will find that your network does not converge quickly after link failures.

In our example, we have attempted to speed up convergence by decreasing the timers. The new hello time is 3 seconds and the hold time is 9 seconds. Before applying this change, you can see that the hold time is 15 seconds:

```
Router1#show ip eigrp neighbors
IP-EIGRP neighbors for process 55
H   Address                 Interface   Hold Uptime   SRTT   RTO  Q  Seq Type
                                        (sec)         (ms)       Cnt Num
0   172.25.2.1              Se0.1         15 00:10:02   16   200  0  549
Router1#
```

This command actually shows the amount of time remaining in the hold time interval. Each time you look at the neighbor table you will see that the router is counting down from the configured hold time. Each time this router receives a hello packet from the specified neighbor router, it resets its hold timer and begins counting down again. If it ever reaches zero, it will reset the neighbor relationship.

If we go to the neighbor router in the example, you can see that the hold time for Router2 counts down from 9 seconds, instead of the default 15:

```
Router2#show ip eigrp neighbors
IP-EIGRP neighbors for process 55
H   Address                 Interface   Hold Uptime   SRTT   RTO  Q  Seq Type
                                        (sec)         (ms)       Cnt Num
1   172.25.2.2              Se0/0.2        9 00:10:50   16   200  0  114
2   172.25.1.7              Fa0/0.1       65 1d22h      15   200  0  377
0   172.22.1.4              Fa0/1         13 1d22h       2   200  0  230
Router2#
```

7.9.4 See Also

Chapter 6; Chapter 8

Top

Recipe 7.10 Enabling EIGRP Authentication

7.10.1 Problem

You want to authenticate your EIGRP traffic to ensure that no unauthorized equipment can affect your routing tables.

7.10.2 Solution

To enable MD5-based EIGRP packet authentication, you must first define a key chain for the encryption, then apply the authentication commands to the interface:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#key chain ORA
Router1(config-keychain)#key 1
Router1(config-keychain-key)#key-string oreilly
Router1(config-keychain-key)#exit
Router1(config-keychain)#exit
Router1(config)#interface Serial0/1
Router1(config-if)#ip authentication mode eigrp 55 md5
Router1(config-if)#ip authentication key-chain eigrp 55 ORA
Router1(config-if)#end
Router1#
```

7.10.3 Discussion

As soon as we configure EIGRP authentication on this router, the neighbor relationship dropped because it failed to authenticate:

```
IP-EIGRP 55: Neighbor 172.25.2.2 (Serial0/0.2) is down: Auth failure
```

To bring this neighbor back up, you have to ensure that both routers use the same authentication keys.

It's important to remember that this is just an authentication system. The routers do not encrypt the routing update packets as they send them through the network. They just authenticate these packets using MD5. This prevents people from either accidentally or maliciously injecting routes into your network. This authentication is often useful in environments where you don't control all of the routers.

You can see from the following debug trace that when the authentication fails, EIGRP simply ignores the routing updates:

```
Router1#debug eigrp packet
```

```
EIGRP Packets debugging is on
    (UPDATE, REQUEST, QUERY, REPLY, HELLO, IPXSAP, PROBE, ACK, STUB, SIAQUERY,
SIAREPLY)
Router1#
Oct  3 01:40:59.704: EIGRP: ignored packet from 172.25.2.2 opcode = 5
(invalid authentication)
```

One of the biggest problems with using this sort of authentication system is that changing the keys can break routing throughout your network. The following example shows a way around this problem. By configuring timed keys, you can roll out a new key throughout your network without disrupting service:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#key chain Mars
Router1(config-keychain)#key 1
Router1(config-keychain-key)#key-string rocket
Router1(config-keychain-key)#accept-lifetime 00:00:00 Jan 1 1993 00:15:00 Nov 1 2002
Router1(config-keychain-key)#send-lifetime 00:00:00 Jan 1 1993 00:00:00 Nov 1 2002
Router1(config-keychain-key)#key 2
Router1(config-keychain-key)#key-string martian
Router1(config-keychain-key)#accept-lifetime 23:45:00 Oct 31 2002  infinite
Router1(config-keychain-key)#send-lifetime 00:00:00 Nov 1 2002  infinite
Router1(config-keychain-key)#end
Router1#
```

In this case, the router will accept the original key string, *rocket*, until 12:15 A.M. on November 1, 2002. It will send this same key string until 12:00 A.M. on the same date. And it will start accepting the new key string, *martian* , at 11:45 P.M. on October 31, 2002. In this way there is a safe 30-minute transition period that you can configure in advance throughout the network. Then, the next day (or whenever it is convenient), you can remove the configuration for the old key string.

The *show key chain* command includes information about all of the configured key chains and the corresponding key strings:

```
Router1#show key chain
Key-chain ORA:
    key 1 -- text "oreilly"
        accept lifetime (always valid) - (always valid) [valid now]
        send lifetime (always valid) - (always valid) [valid now]
Key-chain Mars:
    key 1 -- text "rocket"
        accept lifetime (00:00:00 Jan 1 1993) - (00:15:00 Nov 1 2002)   [valid now]
        send lifetime (00:00:00 Jan 1 1993) - (00:00:00 Nov 1 2002)   [valid now]
    key 2 -- text "martian"
        accept lifetime (23:45:00 Oct 31 2002) - (infinite)
        send lifetime (00:00:00 Nov 1 2002) - (infinite)
Router1#
```

7.10.4 See Also

Recipe 6.14

Top

Recipe 7.11 Logging EIGRP Neighbor State Changes

7.11.1 Problem

You want to log EIGRP neighbor state changes.

7.11.2 Solution

To enable the logging of EIGRP neighbor state changes, use the *eigrp log-neighbor-changes* configuration command:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#router eigrp 55
Router1(config-router)#eigrp log-neighbor-changes
Router1(config-router)#end
Router1#
```

7.11.3 Discussion

When a neighbor relationship is lost, you also lose all of the routing entries for that neighbor. The effects of this lost routing information are often felt throughout the network. Therefore, it can be extremely useful to have a good log of neighbor change events for troubleshooting strange intermittent network problems. This feature also gives you a good way of looking for faults on links that don't have a way of telling you about loss of connectivity.

Two important examples of this are tunnels and LAN extensions. In many cases, when the network breaks and brings down a tunnel, the two tunnel end points are unable to see the problem. Similarly, in a LAN extension service, the two end point routers are both connected to Layer 2 LAN switches that are then bridged to one another through another medium, such as ATM. The problem in this case is that the intermediate network between the switches can break and neither router will see a problem because they are both connected to an active switch port. It's also important to note that EIGRP neighbor relationships can break just because of noisy or congested links.

Whatever the cause, one of the easiest ways to detect a connectivity problem in the hidden network is to configure EIGRP between the routers via this link. In some cases, this will be done on a separate EIGRP process ID number to make it easier to differentiate between normal network topology changes and these hidden network faults. If you log EIGRP neighbor changes and configure the routers to send

their SYSLOG events to a central fault management server (as discussed in Chapter 18), you can get an instant alarm on these types of problems.

The log messages show not only that the neighbors have changed, but they also give you an indication of why they changed state:

```
Oct  2 22:00:38: %DUAL-5-NBRCHANGE: IP-EIGRP 55: Neighbor 172.25.2.1 (Serial0.1)
 is up: new adjacency
Oct  2 22:03:23: %DUAL-5-NBRCHANGE: IP-EIGRP 55: Neighbor 172.25.2.1 (Serial0.1)
 is down: summary configured
Oct  2 22:03:23: %DUAL-5-NBRCHANGE: IP-EIGRP 55: Neighbor 172.25.2.1 (Serial0.1)
 is up: new adjacency
Oct  2 22:04:14: %DUAL-5-NBRCHANGE: IP-EIGRP 55: Neighbor 172.25.2.1 (Serial0.1)
 is down: manually cleared
Oct  2 22:04:19: %DUAL-5-NBRCHANGE: IP-EIGRP 55: Neighbor 172.25.2.1 (Serial0.1)
 is up: new adjacency
Oct  2 22:07:26: %DUAL-5-NBRCHANGE: IP-EIGRP 55: Neighbor 172.25.2.1 (Serial0.1)
 is down: peer restarted
Oct  2 22:07:27: %DUAL-5-NBRCHANGE: IP-EIGRP 55: Neighbor 172.25.2.1 (Serial0.1)
 is up: new adjacency
Oct  2 22:30:06: %DUAL-5-NBRCHANGE: IP-EIGRP 55: Neighbor 172.25.2.1 (Serial0.1)
 is down: holding time expired
Oct  2 22:30:38: %DUAL-5-NBRCHANGE: IP-EIGRP 55: Neighbor 172.25.2.1 (Serial0.1)
 is up: new adjacency
```

In this example, we have shown four different reasons for EIGRP to reset its neighbor relationships. Of these, only the last one, *holding time expired*, is likely to indicate a network fault.

7.11.4 See Also

Chapter 18

Top

Recipe 7.12 Limiting EIGRP's Bandwidth Utilization

7.12.1 Problem

You want to limit the fraction of an interface's bandwidth available to EIGRP for routing updates.

7.12.2 Solution

To modify the fraction of the total bandwidth available to EIGRP, use the *ip bandwidth-percent* configuration command:

```
Router1# configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#interface Serial0.1
Router1(config-subif)#ip bandwidth-percent eigrp 55 40
Router1(config-subif)#end
Router1#
```

7.12.3 Discussion

This example shows how to restrict EIGRP process number 55 to use, at most, 40% of the available capacity of this link. By default, EIGRP will limit its own bandwidth utilization to ensure that it never takes more than 50% of a link's capacity. However, this default isn't always appropriate. Sometimes you need to reduce this fraction to reduce overall congestion. And sometimes the total bandwidth value specified on an interface is not accurate.

For example, in Recipe 7.7 we discussed how to change what the router thinks the interface's bandwidth is. If this value is significantly lower than the real physical bandwidth of the interface, you might want to increase the fraction that EIGRP can use. This can help to improve network convergence times when EIGRP suddenly needs to exchange a large amount of routing information.

In the following example, we have manually reduced bandwidth value on this interface to 32Kbps. Since this is much less than the true value, we have then increased the fraction that EIGRP can use to 200%, bringing it up to a maximum of 56Kbps, which would be the default for a real 128kbps circuit:

```
Router1# configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#interface Serial0.1
Router1(config-subif)#bandwidth 32
Router1(config-subif)#ip bandwidth-percent eigrp 55 200
```

```
Router1(config-subif)#end
Router1#
```

7.12.4 See Also

Recipe 7.7

Top

Recipe 7.13 EIGRP Stub Routing

7.13.1 Problem

You want to stabilize your network by sending smaller routing tables out to stub branches and reducing the scope of EIGRP queries.

7.13.2 Solution

To enable stub routing, use the *eigrp stub* configuration command:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#router eigrp 55
Router1(config-router)#eigrp stub
Router1(config-router)#end
Router1#
```

7.13.3 Discussion

This feature became available starting in IOS 12.0(15)S. It is most commonly used in hub-and-spoke network designs, in which a remote router connects to the rest of the network through only one or two central routers, and the remote router is the only connection for a small number of LAN segments. In general, you would configure the central routers in this case to send only a default route, as discussed in Recipe 7.2, Recipe 7.5, and Recipe 7.10.

In situations where a route suddenly goes away, every router will ask all of its neighbors if they have a path to that remote network by default. However, it is never going to be fruitful to ask these remote branch routers if they can reach the missing network. If these stub routers are reachable at all, they will already be exchanging information about the few networks that they can access. No trick of topology will allow them to find the missing route if the central routers don't have it. So the EIGRP stub feature disables these queries. This should help to improve overall network stability. And, in particular, this feature could significantly improve the stability of hub-and-spoke WANs:

```
Router2#show ip eigrp neighbors detail
IP-EIGRP neighbors for process 55
H   Address                 Interface   Hold Uptime   SRTT   RTO  Q  Seq Type
                                        (sec)         (ms)       Cnt Num
1   172.25.2.2              Se0/0.2        6 00:15:57  787  4722  0  148
   Version 12.2/1.2, Retrans: 0, Retries: 0
```

```
   Stub Peer Advertising ( CONNECTED SUMMARY ) Routes
2   172.25.1.7              Fa0/0.1       70 1w0d       12   200  0  405
   Version 12.0/1.0, Retrans: 1, Retries: 0
0   172.22.1.4              Fa0/1         12 1w0d        1   200  0  258
   Version 12.2/1.2, Retrans: 2, Retries: 0
Router2#
```

The *eigrp stub* command can take four different keywords:

Receive-only

> The router becomes a receive-only neighbor. This router will not share its routing information with its neighbors.

Connected

> This router will only advertise connected networks. Note that you must configure the appropriate network statements for these connected networks, or alternatively use the *redistribute connected* command.

Static

> The router will advertise static routes. Note that with this option you must also configure the *redistribute static* command.

Summary

> The router will advertise summary routes. This function is enabled by default. Refer to Recipe 7.10 for details on route summarization.

7.13.4 See Also

Recipe 7.2; Recipe 7.5; Recipe 7.10

Top

Recipe 7.14 Route Tagging

7.14.1 Problem

You want to tag specific routes to prevent routing loops while mutually redistributing routes between two routing protocols.

7.14.2 Solution

This example shows how to tag external routes in EIGRP:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#ip route 0.0.0.0 0.0.0.0 172.25.1.1
Router1(config)#access-list 7 permit 0.0.0.0
Router1(config)#route-map TAGGING permit 10
Router1(config-route-map)#match ip address 7
Router1(config-route-map)#set tag 5
Router1(config-route-map)#exit
Router1(config)#router eigrp 55
Router1(config-router)#redistribute static route-map TAGGING
Router1(config-router)#end
Router1#
```

The external route is static here, but the same technique applies to routes learned through other routing protocols.

7.14.3 Discussion

You can only tag routes that EIGRP has learned from another routing protocol. As we saw when talking about route tags with RIP, EIGRP does not use these tags directly, it only distributes them. You would use these tags at network boundaries when redistributing routes into another routing process.

For instance, if our EIGRP network was being used as a transit network between two other routing protocols we could tag routes learned from the first external network. We could then redistribute only the first network's routes into the second external network. Similarly, we could redistribute only the second network's routes into the first network. In this way we could ensure that the external networks use the EIGRP network for only transit, and prevent them from reaching anything internal.

As discussed in Chapter 6, RIP Version 2 supports 16-bit tags, which gives it a range from 0 to 65,535. EIGRP and OSPF use 32-bit tags for a range from 0 to 4,294,967,295. These tags are purely internal, of course, so there is no interoperability problem. It is unlikely that you will need 4 billion tags, but this expanded range can be useful because you can map the 32-bit tags to IP addresses as a mnemonic for the external network information:

```
Router1#show ip eigrp topology 0.0.0.0
IP-EIGRP (AS 55): Topology entry for 0.0.0.0/0
  State is Passive, Query origin flag is 1, 1 Successor(s), FD is 28160
  Routing Descriptor Blocks:
  0.0.0.0, from Rstatic, Send flag is 0x0
      Composite metric is (28160/0),  Route is External
      Vector metric:
        Minimum bandwidth is 100000 Kbit
        Total delay is 100 microseconds
        Reliability is 255/255
        Load is 1/255
        Minimum MTU is 1500
        Hop count is 0
      External data:
         Originating router is 172.25.25.1 (this system)
        AS number of route is 0
        External protocol is Static, external metric is 0
        Administrator tag is 5 (0x00000005)
        Exterior flag is set
  0.0.0.0 (Null0), from 0.0.0.0, Send flag is 0x0
      Composite metric is (28160/0), Route is Internal
      Vector metric:
        Minimum bandwidth is 100000 Kbit
        Total delay is 100 microseconds
        Reliability is 255/255
        Load is 1/255
        Minimum MTU is 1500
        Hop count is 0
      Exterior flag is set
Router1#
```

The following is a simple example in which we redistribute from EIGRP into OSPF only those routes that have a route tag value of 5. Presumably this tag was set at another network boundary:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#router ospf 11
Router1(config-router)#redistribute eigrp 55 route-map TAGOUT
Router1(config-router)#exit
Router1(config)#route-map TAGOUT permit 10
Router1(config-route-map)#match tag 5
Router1(config-route-map)#route-map TAGOUT deny 20
Router1(config-route-map)#end
Router1#
```

7.14.4 See Also

Chapter 6

Top

Recipe 7.15 Viewing EIGRP Status

7.15.1 Problem

You want to check the status of EIGRP on the router.

7.15.2 Solution

There are several useful commands for looking at EIGRP status. As we have seen throughout this chapter, the *show ip protocols* command displays a wealth of useful information:

```
Router1#show ip protocols
```

You can look at a routing table of only those routes that were learned via EIGRP as follows:

```
Router1#show ip route eigrp
```

Another extremely useful EIGRP command displays a table of all of the adjacent EIGRP routers:

```
Router1#show ip eigrp neighbors
```

You can see information about the interfaces that exchange routing information using EIGRP with this command:

```
Router1#show ip eigrp interfaces
```

Finally, you can view the EIGRP topology database as follows:

```
Router1#show ip eigrp topology
```

7.15.3 Discussion

The precise output of the *show ip protocols* command varies depending on what features are enabled. However, we have shown several examples of different output throughout this chapter:

```
Router1#show ip protocols
Routing Protocol is "eigrp 55"
  Outgoing update filter list for all interfaces is not set
  Incoming update filter list for all interfaces is not set
    Serial0.1 filtered by (prefix-list) Inbound
  Default networks flagged in outgoing updates
  Default networks not accepted from incoming updates
```

```
  EIGRP metric weight K1=1, K2=0, K3=1, K4=0, K5=0
  EIGRP maximum hopcount 100
  EIGRP maximum metric variance 1
  Redistributing: static, eigrp 55
  Automatic network summarization is not in effect
  Maximum path: 4
  Routing for Networks:
    172.25.2.2/32
    172.25.0.0
    192.168.20.0
  Routing Information Sources:
    Gateway         Distance      Last Update
    (this router)         90      5d23h
    172.25.2.1            90      00:03:32
  Distance: internal 90 external 170
Router1#
```

The standard command to view the IP routing table is *show ip route*. However, this will show you all of the IP routes, including static routes, connected routes, routes learned through other protocols, and EIGRP routes. If you just want to see the EIGRP routes, you can add the keyword*eigrp* to this command:

```
Router1#show ip route eigrp
D*EX 0.0.0.0/0 [170/91942912] via 172.25.2.1, 00:04:29, Serial0.1
Router1#
```

One of the most useful commands when troubleshooting EIGRP problems looks at the neighbor table:

```
Router1#show ip eigrp neighbors
IP-EIGRP neighbors for process 55
H   Address                 Interface   Hold Uptime   SRTT   RTO  Q  Seq Type
                                        (sec)         (ms)       Cnt Num
1   172.25.2.2              Se0/0.2        7 00:25:16  641  3846  0  148
2   172.25.1.7              Fa0/0.1       80 1w0d       17   200  0  406
0   172.22.1.4              Fa0/1         12 1w0d        3   200  0  259
Router1#
```

There are several important pieces of information in this list. Obviously, it's useful to look at the IP addresses and interfaces. But it can also be extremely useful to look at the uptime. In this case, two of these neighbors have been up and stable for a week, but the third was reset 25 minutes ago. The router will sort this output so that the most recent neighbors are at the top. This gives you an immediate way to see which neighbors might have problems.

Also useful in this output is the "Q" column. This column tells you how many EIGRP packets are currently queued for the specified neighbor. If the router is consistently queueing EIGRP packets, there may be a congestion or queueing problem with this interface.

If you think you see EIGRP congestion or performance problems like this, it can be useful to look at the interfaces in more detail:

```
Router1#show ip eigrp interfaces
IP-EIGRP interfaces for process 55

                   Xmit Queue   Mean   Pacing Time   Multicast    Pending
Interface    Peers  Un/Reliable  SRTT   Un/Reliable   Flow Timer   Routes
Fa0/0.1        1        0/0        17       0/10          50           0
Lo0            0        0/0         0       0/10           0           0
Fa0/1          1        0/0         3       0/10          50           0
Se0/0.2        1        0/0       641       0/15        3163           0
Router1#
```

This command shows useful information, such as how many peers there are on each interface. It also tells you more about any possible queueing issues, breaking out exactly how many routes are still pending.

Another useful command when debugging EIGRP problems is the *show ip eigrp toplogy* command. This command gives a view of the EIGRP topology table. The topology table is useful because it often includes information about routes that EIGRP has received, but that the router isn't using for whatever reason. For example, if there is a similar route with a better administrative distance, such as a static route, the *show ip route* command will indicate only the static route. The *show ip eigrp toplogy* command allows you to look through the whole EIGRP topology table to see exactly why the other route is better:

```
Router1#show ip eigrp topology
IP-EIGRP Topology Table for AS(55)/ID(172.25.25.1)

Codes: P - Passive, A - Active, U - Update, Q - Query, R - Reply,
       r - reply Status, s - sia Status

P 0.0.0.0/0, 1 successors, FD is 28160, tag is 5
         via Rstatic (28160/0)
         via Summary (28160/0), Null0
P 10.2.2.0/24, 1 successors, FD is 156160
          via 172.22.1.4 (156160/128256), FastEthernet0/1
P 10.1.1.0/30, 1 successors, FD is 3845120
         via Connected, Serial0/1
P 192.168.10.0/24, 1 successors, FD is 28160, tag is 5
        via Rstatic (28160/0)
P 192.168.30.0/24, 1 successors, FD is 156160
         via 172.22.1.4 (156160/128256), FastEthernet0/1
P 192.168.20.0/24, 1 successors, FD is 2195456
         via 172.25.2.2 (2195456/281600), Serial0/0.2
P 172.25.25.6/32, 1 successors, FD is 156160
         via 172.25.1.7 (156160/128256), FastEthernet0/0.1
P 172.25.25.1/32, 1 successors, FD is 128256
        via Connected, Loopback0
P 172.25.25.2/32, 1 successors, FD is 2297856
         via 172.25.2.2 (2297856/128256), Serial0/0.2
P 172.25.1.0/24, 1 successors, FD is 28160
```

```
          via Connected, FastEthernet0/0.1
P 172.25.2.0/30, 1 successors, FD is 2169856
          via Connected, Serial0/0.2
P 172.22.1.0/24, 1 successors, FD is 28160
          via Connected, FastEthernet0/1
Router1#
```

The EIGRP topology table also shows the successors for each route. The successor is the route that will be installed in case the better one goes away.

Top

Chapter 8. OSPF

Top

Introduction

Open Shortest Path First (OSPF) is a popular routing protocol for IP networks for several key reasons. It is classless, offering full CIDR and VLSM support, it scales well, converges quickly, and guarantees loop free routing. It also supports address summarization and the tagging of external routes, similar to EIGRP. For networks that require additional security, you can configure OSPF routers to authenticate with one another to ensure that unauthorized devices can't affect routing tables.

Perhaps the most important reasons for OSPF's popularity are that it is both an open standard and a mature protocol. Virtually every vendor of routing hardware and software supports it. This makes it the routing protocol of choice in multivendor enterprise networks. It is also frequently found in ISP networks for the same reasons.

But, for all of these benefits, OSPF is also considerably more complicated to set up than EIGRP or RIP. Unlike EIGRP, which can be readily retrofitted into almost any existing network, your network has to be designed with OSPF in mind if you want it to scale well. For more information on OSPF network design, refer to Designing Large-Scale LANs (O'Reilly). You can find more information about the protocol itself in IP Routing (O'Reilly). The remainder of this section is intended only to serve as a reminder to readers who are already familiar with OSPF.

OSPF is currently in its second version, which is documented in RFC 2328. It uses a large, dimensionless metric on every link (also equivalently called a "cost"), with a maximum value of 65,535. It is important to remember that OSPF doesn't add these metrics the same way that RIP and EIGRP do. In those protocols, each router updates the total metric as it passes the route on to the next router. However, in OSPF, the routers distribute the individual link costs to one another. The maximum cost for an individual link, then, is 65,535, but the RFC does not specify a maximum total path cost. Any given path through an OSPF network can include many high-cost links, but still be usable. This is quite different from RIP, for example, where a few high-cost links along a path can make the entire path unusable.

This 16-bit OSPF per-link metric, while significantly larger than the simple hop-count metric used in RIP, is much smaller than EIGRP's 32-bit metric. So many of the metric manipulation techniques we discussed for EIGRP in Chapter 7 do not work in OSPF. The smaller metric sometimes means that you have to exercise care in how you define the costs of each link. We discuss this issue in more detail in Recipe 8.3.

Like EIGRP, OSPF routers only start to exchange routing information after they have established a neighbor relationship. However, unlike EIGRP, OSPF routers don't actually exchange routing tables directly. Instead, they exchange Link State Advertisements (LSAs), which describe the states of

different network links. Each router then obtains an accurate image of the current topology of the network, which it uses to build its routing tables. If you group the routers into areas, as we will discuss in a moment, every router in each area sees the same LSA information, which guarantees that all of the routing tables are compatible with one another.

The OSPF protocol operates directly at the IP layer using IP protocol number 89, without an intervening transport layer protocol such as UDP or TCP. Devices exchange OSPF information using multicast packets that are confined to the local segment. OSPF actually uses two different multicast IP addresses: all OSPF routers use `224.0.0.5`, and Designated Routers (DRs) use `224.0.0.6`.

A DR is basically a master router for a network segment. This is only relevant when there are several OSPF routers on a multiple access medium, such as an Ethernet segment. In this case, to avoid the scaling problems of establishing a mesh of neighbor relationships between all of the routers on the segment, one router becomes the DR for the segment. Then all of the other routers talk to the DR. Each segment also elects a Backup Designated Router (BDR) in case the DR fails.

One of the most important features of OSPF is the concept of an area. This is also partly what makes OSPF more difficult to configure. An OSPF network can be broken up into areas that are connected by Area Border Routers (ABRs). Routing information can then be summarized at the ABR before being passed along to the next area. This means that routers in one area don't need to worry about the LSA information from routers in other areas, which drastically improves network stability and convergence times. It also reduces the memory and CPU required to support OSPF on the routers.

For OSPF to work well, you need to allocate your IP addresses appropriately among the areas. In particular, you want to be able to summarize the routes for an area when you pass this information along to the next area. The summarization doesn't need to reduce perfectly to a single route for each area, but the fewer LSAs you need to pass between areas, the better OSPF will scale.

Each area has a 32-bit identifier number, which is often represented in dotted decimal notation, similar to IP addresses. Every OSPF network should have an Area 0 (or`0.0.0.0`), and every ABR must be a member of Area 0. This enforces a hierarchical design model for OSPF networks. The one exception to this rule happens in a network with only one area. In this case you can actually give this area any number, but we don't recommend doing so because it could cause serious problems if you ever need to partition the network into areas later. The only time this becomes relevant is when a network failure isolates one area from the rest of the network. In this case, the isolated area can continue working as normal internally.

You can get around this strict design requirement of having all areas connected only through Area 0 by using OSPF virtual links. These are essentially little more than IP tunnels. You can use virtual links to ensure that every ABR connects to Area 0, even if one or more of them are not physically connected to Area 0. However, we should stress that we do not recommend using virtual links except as a temporary measure—perhaps while migrating your network to a new architecture or while merging two networks.

The OSPF protocol defines several different LSA types. We will briefly review these different types before discussing the area types, because it will help you to understand what is going on in these different area types. The standard LSA types are summarized in Table 8-1.

Table 8-1. LSA types

LSA type	Name	Description
1	Router-LSA	A Router-LSA includes information about the link states of all of a router's interfaces. These LSAs are flooded throughout the area, but not into adjacent areas.
2	Network-LSA	On NBMA and broadcast-capable network segments, the DR originates Network-LSAs. The Network-LSA describes the routers that are connected to this broadcast or NBMA segment. Network-LSAs are flooded throughout the area, but not into adjacent areas.
3	Summary-LSA	ABR routers originate Summary-LSAs to describe inter-area routes to networks that are outside of the area but inside of the AS. They are flooded throughout an area. Type 3 LSAs are used for routes to networks.
4	Summary-LSA	Type 4 LSAs are similar to Type 3 LSAs, except that they are used for routes to ASBR routers.
5	AS-External-LSA	ASBR routers originate Type 5 LSAs to describe routes to networks that are external to the AS. Type 5 LSAs are flooded throughout the AS.
6	MOSPF-LSA	Type 6 LSAs are used for carrying multicast routing information with MOSPF. Cisco routers do not currently support Type 6 LSAs.
7	NSSA-External-LSA	Type 7 LSAs are originated by ASBRs in an NSSA area. They are similar to Type 5 LSAs, except that they are only flooded throughout the NSSA area. When Type 7 LSAs reach the ABR, it translates them into Type 5 LSAs and distributes them to the rest of the AS.

There are several different types of OSPF areas. They are differentiated by how they summarize information into and out of the area. The other important difference between area types concerns whether or not they can be used for transit between other parts of the network. Transit means that the area can allow packets to pass through the area on their way to another area or another network. Any router that connects OSPF to another network or a different routing protocol is called an Autonomous System Boundary Router (ASBR). Clearly, to be useful, any area that includes an ASBR needs to allow transit.

The first important type of area is the backbone area, which is used by Area 0. This area is special because it can always act as a transit area between other areas, between this OSPF autonomous system and external networks, or even between external networks.

A regular area connects to the backbone area. Every router in a regular area sees the Type 1 and 2 LSAs for every other router in the area. They use Type 3 LSAs to learn how to route to destinations in other areas, and Type 4 and 5 LSAs when routing to destinations outside of the OSPF network. All of the other types of areas that we will describe are modifications of a regular area.

The third area type is called a stub area. Stub areas see detailed routing information on all other areas, but only summary information about networks outside of the AS. The ABR sends Type 3 LSA packets to summarize this information. The ABR connecting to a stub area summarizes routes to external networks outside of the AS. All external routes are reduced to a single summary. This is important because it means that you cannot make connections to external networks via a stub area. It also means that, if your network is essentially all one big AS (perhaps with a default route to the Internet), there is no advantage to using a stub area. Stub areas are most useful when there are many external routes, so summarizing them saves router resources.

In terms of LSA types, the distinguishing factor for a stub area is that the ABR will not send any Type 5 LSAs into this area.

Fourth is the totally stub area. Totally stub areas, also called "stub no-summary areas," summarize not only external routes, but also routes from other areas (inter-area routes). Routers in this type of area only see routing information local to their area, plus a default route pointing to the ABR, from which they can reach all other areas and all other networks. The ABR accomplishes this by preventing all Type 3, 4, and 5 LSA messages, except for the default summary route, which it transmits as a single Type 3 LSA message.

As with regular stub areas, you cannot make connections to external networks through totally stub areas using redistribution into OSPF.

Totally stub areas are clearly useful in WAN situations where the overhead of maintaining and updating a large link state database is both onerous and unnecessary. The only problem with totally stub areas is that this is essentially a Cisco invention. Some other vendors have added support for this area type, but it is not universally supported, so you might have problems implementing it in a multivendor network. But, as long as you use Cisco ABR routers, the other routers inside of a totally stub area won't know that anything special has happened to their routing information, so the non-ABR routers can be non-Cisco devices.

Not so stubby areas (NSSA) are defined in RFC 1587. This is a variant of the stub area that is able to connect to external networks. It accomplishes this by introducing a new LSA type (LSA Type 7) that is used within the area to carry external routes that originate with ASBRs connected to this area. The ABR summarizes only those external routes that are received from other areas, and therefore reached through

the ABR. External routes from ASBRs inside the area are not summarized. In order to pass the internally generated external routes to the rest of the network, the ABR translates these Type 7 LSAs into the more conventional Type 5 LSAs before relaying this information into Area 0.

The result is that you can use NSSA areas to connect to external networks. This is extremely important to remember, because even a simple redistributed static route is considered an external route. If you want external routes to be available for the rest of the network, then NSSA is a good way to handle them. NSSA is an open standard part of the OSPF protocol, so most of the router vendors who implement OSPF include NSSA support.

Finally, another useful Cisco adaptation is the totally stubby not so stubby area type. This comical sounding name belies an extremely useful feature. This area type combines the best of NSSA and totally stub areas by summarizing information from all other areas, but handling external routes like NSSA. It allows you to summarize internal routes from other areas while still allowing you to put an ASBR inside of the area.

As with the totally stub area, the ABR connecting to a totally stubby NSSA area prevents all Type 3, 4 and 5 LSAs. And, like an NSSA, it uses Type 7 LSA messages to carry external routes from ASBR routers inside of the area. So the totally stubby NSSA area can be used as a transit area to an external network, but it can also benefit from summarization of inter-area routes.

In many networks, the number of external routes is relatively small, while there are many internal (inter-area) routes. So it is actually much more important to summarize the internal routes in these cases. But the totally stub area type that allows this inter-area route summarization doesn't allow you to connect to the external networks. The totally stubby NSSA area type is ideal when you need to connect to an external network through an area that you would really prefer to keep stubby for performance and scaling reasons.

Another important concept in OSPF involves how it exchanges routing information with external autonomous systems. OSPF defines two different types of external routes. The only difference between them is in the way that OSPF calculates their costs. The cost of a Type 1 external route is the sum of the external metric plus the internal cost to reach the ASBR. The cost of a Type 2 external route is just the external metric cost. OSPF does not add in the cost to reach the ASBR for Type 2 external routes.

When making routing decisions, OSPF prefers Type 1 to Type 2 external routes. So, for example, you can use Type 1 external routes to ensure that every internal router selects the closest ASBR that connects to a particular external network. But you might want to also set up a backup ASBR that injects Type 2 routes. The internal routers will then prefer the Type 1 routes if they are present.

Top

Recipe 8.1 Configuring OSPF

8.1.1 Problem

You want to run OSPF on a simple network.

8.1.2 Solution

You can enable OSPF on a router by defining an OSPF process and assigning an address range to an area as follows:

```
Router5#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router5(config)#router ospf 87
Router5(config-router)#network 0.0.0.0 255.255.255.255  area 0
Router5(config-router)#end
Router5#
```

8.1.3 Discussion

The first line in this configuration example defines the OSPF process:

```
Router5(config)#router ospf 87
```

The OSPF process number (87) doesn't propagate outside of the area, so you can actually use any value here. In fact, the OSPF process number doesn't even propagate outside of the router; you can use a different value for every router in an AS. Note that this is different from EIGRP, where every router in the AS must have the same process number. The process number can take any value between 1 and 65,535.

The network statement in this example takes the simplest possible approach to defining areas by putting every interface on the router into Area 0:

```
Router5(config-router)#network 0.0.0.0 255.255.255.255  area 0
```

The first two arguments of the network statement are an IP address and a corresponding set of wildcard bits. In this case, `0.0.0.0  255.255.255.255` matches every possible IP address, so every interface on this router is assigned to Area 0.

In this example, we have defined the area using a single number. You can define area numbers to be

anything between 0 and 4,294,967,295. You can also use dotted decimal notation for areas, in which case you would write Area 0 as `0.0.0.0`.

OSPF treats the area number internally as a 32-bit field regardless of whether you choose to represent it as a single number or by using dotted decimals. You could mix and match if you wanted to, but we recommend picking whichever approach better suits your network and sticking to it. It is far too easy to get confused when translating between the two formats.

For small networks it is generally easier to remember single numbers. But for larger networks it is often easier to work with dotted decimal notation, perhaps developing some locally meaningful mnemonic to help give the area numbers significance. For example, in an international WAN, you might want to make the first octet stand for the country, the second for state or province, the third for city, and the fourth for the actual area. Or you might want to make the area numbers the same as the summary IP addresses for each area. The only special area number is Area 0, which is the routing backbone of the network and must connect to all other areas directly, as we discussed in the introduction to this chapter.

If you have more than one *network* statement, the order becomes important. In the following example, the last line matches all IP addresses and assigns them to Area 0. But, because this line comes last, it only picks up any addresses that are not captured by either of the lines above it. However, if we had written this line first, then all of the interfaces would wind up in Area 0:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#router ospf 55
Router1(config-router)#network 10.0.0.0 0.255.255.255  area 2
Router1(config-router)#network 172.20.0.0 0.0.255.255  area 100
Router1(config-router)#network 0.0.0.0  255.255.255.255 area 0
Router1(config-router)#end
Router1#
```

The *show ip protocols* command lists useful information about the OSPF configuration:

```
Router1#show ip protocols
Routing Protocol is "ospf 55"
  Outgoing update filter list for all interfaces is not set
  Incoming update filter list for all interfaces is not set
  Router ID 172.25.25.1
  It is an area border and autonomous system boundary router
  Redistributing External Routes from,
  Number of areas in this router is 3. 3 normal 0 stub 0 nssa
  Maximum path: 4
  Routing for Networks:
    10.0.0.0 0.255.255.255 area 2
    172.20.0.0 0.0.255.255 area 100
    0.0.0.0 255.255.255.255 area 0
  Routing Information Sources:
    Gateway         Distance      Last Update
    192.168.30.1         110      00:01:30
```

```
    172.25.25.6           110      16:44:07
    172.25.25.1           110      00:01:30
    172.25.25.2           110      00:01:30
    172.25.1.7            110      00:01:30
    172.25.1.1            110      16:56:15
  Distance: (default is 110)
Router1#
```

If you want more control, you can specify individual interface IP addresses with a wildcard mask of `0.0.0.0` to match only one address at a time:

```
Router5#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router5(config)#router ospf 87
Router5(config-router)#network 172.25.1.7 0.0.0.0  area 101
Router5(config-router)#network 172.25.25.6 0.0.0.0  area 102
Router5(config-router)#end
Router5#
```

This can be useful when your addresses don't summarize well, which can be the case when you are changing your network architecture or merging two networks.

Top

Recipe 8.2 Filtering Routes in OSPF

8.2.1 Problem

You want to apply a filter so that OSPF populates only certain routes into the routing table.

8.2.2 Solution

You can filter inbound routes to prevent the router from putting them in its routing table:

```
Router5#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router5(config)#access-list 1 deny 172.20.10.0
Router5(config)#access-list 1 permit any
Router5(config)#router ospf 87
Router5(config-router)#distribute-list 1 in Ethernet0
Router5(config-router)#end
Router5#
```

The OSPF algorithm requires that every router in an area receive all of the LSAs for that area, so you cannot filter outbound routing information:

```
Router5#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router5(config)#router ospf 87
Router5(config-router)#distribute-list 1 out Ethernet0
% Interface not allowed with OUT for OSPF
Router5(config-router)#end
Router5#
```

8.2.3 Discussion

It's important to remember that, unlike EIGRP and RIP, OSPF uses a link state rather than a distance vector algorithm. One place where this difference becomes clear is in route filtering. At a minimum, every router in an area must see the LSAs for every other router in the same area. Depending on the type of area, it may also see summary LSAs representing routing information from other areas or ASes. These LSA packets are flooded throughout the area, with each router forwarding LSA information on to any downstream devices. Every router then separately computes the best routing table based on this link state information.

If you prevented a router from forwarding some of the LSA information, its downstream routers would

not have a full link-state database, and consequently wouldn't be able to generate an accurate routing table.

Therefore, it is not possible to do the kind of route filtering that we discussed for RIP and EIGRP in Chapter 6 and Chapter 7. The only filtering we can do is to prevent a router from installing a route learned via OSPF into its routing table. This way, the link state database remains intact on every router in the area. If you really want to break up the forwarding of LSA information, subdivide the area.

You can see the effect of the inbound filter by looking at the routing table both before and after applying the filter. Before the inbound filter is enabled, you can see that the route is there:

```
Router5#show ip route 172.20.10.0
Routing entry for 172.20.10.0/24
  Known via "ospf 87", distance 110, metric 84, type inter area
  Redistributing via ospf 87
  Last update from 172.25.1.5 on Ethernet0, 00:00:07 ago
  Routing Descriptor Blocks:
  * 172.25.1.5, from 172.25.25.1, 00:00:07 ago, via Ethernet0
      Route metric is 84, traffic share count is 1
Router5#
```

Then, after we apply the filter, the route is gone:

```
Router5#show ip route 172.20.10.0
% Subnet not in table
Router5#
```

8.2.4 See Also

Chapter 6; Chapter 7

Top

Recipe 8.3 Adjusting OSPF Costs

8.3.1 Problem

You want to change the OSPF link costs.

8.3.2 Solution

The *auto-cost reference-bandwidth* command allows you to change the reference bandwidth that OSPF uses to calculate its metrics:

```
Router5#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router5(config)#router ospf 87
Router5(config-router)#auto-cost reference-bandwidth 1000
Router5(config-router)#end
Router5#
```

You can also adjust the OSPF cost of a single interface with the *ip ospf cost* configuration command:

```
Router5#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router5(config)#interface Ethernet0
Router5(config-if)#ip ospf cost 31
Router5(config-if)#end
Router5#
```

8.3.3 Discussion

The custom in OSPF networks is to make the link cost inversely proportional to the bandwidth of a link. This isn't required, but it is common, and it is the default behavior for Cisco routers. The reference bandwidth defines the link speed that has an OSPF cost of 1. By default, the reference bandwidth is 100Mbps.

However, if you have faster links in your network (such as gigabit Ethernet or even OC-3 connections), OSPF can't give these links a better cost than 1. So you should set the reference bandwidth to at least as high as the fastest link in your network. In fact, you may want to set this value higher than the bandwidth of your fastest link to ensure that you don't have to reconfigure your whole network when you eventually upgrade some of your core links.

It is important to set the same reference bandwidth on all routers in an area, and preferably throughout the entire network. Recall that OSPF allows every router to calculate its own routing table based on the LSAs that they receive. So, they must all agree on the relationship between costs and bandwidth. Suppose you set the reference bandwidth differently on two routers, causing them to advertise different link costs for their Ethernet interfaces. This could cause seriously strange routing patterns as OSPF will try to avoid using the higher-cost links. It may decide, for example, that a FastEthernet interface on one router is faster than a Gigabit Ethernet interface on the other router.

But there is another interesting problem with the way OSPF calculates its metrics. Suppose your network includes one or more Gigabit Ethernet links in the core and a WAN that provides 9.6Kbps dial backup for the most remote branches. This means that the fastest link is over 100,000 times as fast as the slowest. When this happens, the router will simply apply the maximum link cost of 65,535.

The problem is that the OSPF metric is only 16 bits long, giving it a maximum per-link cost value of 65,535. So, if your fastest link used the emerging 10Gbps Ethernet standard, and you set the cost of this link to 1, then a relatively common 56Kbps serial link would need to have a cost of 178,571. Since this is not possible, OSPF would have to use the maximum link cost of 65,535. This in itself is not a problem. The problem is that your 128Kbps circuits would also need to have the same cost. This could cause some very poor routing patterns.

We suggest using an alternate costing strategy to avoid this problem. The idea is that the cost of a link doesn't actually have to be 10 times as high just because the link is 1/10 as fast. In fact, this default behavior implies that it is better to go through a succession of 10 FastEthernet links rather than use a single Ethernet, which is probably not true in most cases. So a useful alternative strategy is to use the square root of the bandwidth instead of the bandwidth when calculating the link cost. The result of this strategy is shown in Table 8-1Table 8-1.

Table 8-2. Suggested OSPF costs for different media

Medium	Nominal bandwidth	Default Cost	Changing reference bandwidth to 10Gbps	Cost with 1/square root model
9.6kbps line	9.6kbps	10,416	1,041,666	1,020
56kbps line	56kbps	1,785	178,571	422
64kbps line	64kbps	1,562	156,250	395
T1 circuit	1.544Mbps	64	6,476	80
E1 circuit	2.048Mbps	48	4,882	69
T3 circuit	45Mbps	2	222	14

Medium	Nominal bandwidth	Default Cost	Changing reference bandwidth to 10Gbps	Cost with 1/square root model
4Mbps Token Ring	4Mbps	25	2,500	50
16Mbps Token Ring	16Mbps	6	625	25
Ethernet	10Mbps	10	1,000	31
Fast Ethernet	100Mbps	1	100	10
Gigabit Ethernet	1Gbps	1	10	3
10 Gigabit Ethernet	10Gbps	1	1	1

As you can see in Table 8-1, if you use the default costs, then the three fastest links all wind up with a cost of 1. Changing the reference bandwidth to 10Gbps, however, produces impossibly large metrics for the three slowest links (the router will assign them all a link cost of 65,535). So, no matter what reference bandwidth you use, if you retain the default 1/bandwidth cost mode, you will need to manually adjust either the fastest or slowest several link costs.

The second example shows how to change the OSPF cost for a single interface:

```
Router5(config)#interface Ethernet0
Router5(config-if)#ip ospf cost 31
```

If you change the OSPF cost of a link like this, make sure that all of the other routers that connect to this same network segment use the same cost, or strange routing will result. Further, it is usually a good practice to ensure that all of the links of a particular type have the same cost throughout the network, or at least throughout the area. So, if we set the OSPF cost for 10Mbps Ethernet to 31 on this router, we should do the same on the other routers as well.

There are times when you will want to make sure that one particular link is favored, regardless of its actual bandwidth. In this case, you can give this link a special OSPF cost value. However, you will generally want to be careful that all of the devices on this link agree on the cost.

You could also achieve a similar effect by just adjusting the *bandwidth* statements on each interface, but this will have other consequences as well. It will affect any other routing protocols that you might also be running. It also means that an SNMP query of the interface speed will give an incorrect value, which could confuse network management software. We recommend the direct approach of manually setting the OSPF cost because it is more clear what you are doing and why. This becomes most important when somebody else comes along and wants to change the router configuration. If your cost

scheme is not clear, you could cause serious support problems for your successor.

Top

Recipe 8.4 Creating a Default Route in OSPF

8.4.1 Problem

You want to propagate a default route within an OSPF network.

8.4.2 Solution

To propagate a default route with OSPF, use the *default-information originate* configuration command:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#ip route 0.0.0.0 0.0.0.0  172.25.1.1
Router1(config)#router ospf 55
Router1(config-router)#default-information originate metric 30 metric-type 1
Router1(config-router)#end
Router1#
```

8.4.3 Discussion

Unlike RIP and EIGRP, you cannot create a default route in OSPF by simply redistributing a static route. Even if there is a default route in the routing table, Cisco's OSPF implementation will not forward it to the rest of the network by default. This is because OSPF uses a link state algorithm that keeps track of links rather than routes. So summary routes are very special elements in OSPF, and it's important to be careful when distributing them. The default route, `0.0.0.0/0`, is the ultimate summary of summaries, and it has the potential to cause serious confusion if it isn't handled properly.

Cisco forces you to be sure that you really want to source a default route into OSPF by requiring you to specifically enable it with the *default-information originate* command. This command also allows you to specify precisely the metric of this default route and, since a default route is implicitly external to the AS, the type of external route. This has the added advantage of giving finer granularity of control over default route propagation.

You can look at the external routes in the OSPF database with the following command:

```
Router1#show ip ospf database external

            OSPF Router with ID (172.25.25.1) (Process ID 55)

                Type-5 AS External Link States
```

```
  LS age: 163
  Options: (No TOS-capability, DC)
  LS Type: AS External Link
  Link State ID: 0.0.0.0 (External Network Number )
  Advertising Router: 172.25.25.1
  LS Seq Number: 80000002
  Checksum: 0x18E6
  Length: 36
  Network Mask: /0
        Metric Type: 1 (Comparable directly to link state metric)
        TOS: 0
        Metric: 30
        Forward Address: 0.0.0.0
        External Route Tag: 55

Router1#
```

In this example, you can see that the default route is advertised by the `172.25.25.1` router with a metric of 30 and a metric type of 1. The metric type in this case refers to whether this route is considered by OSPF to be a Type 1 or 2 external route. It is a Type 1 route—we configured it this way with the *default-information* command:

```
Router1(config-router)#default-information originate metric 30 metric-type 1
```

As we mentioned in the introduction to this chapter, the cost of a Type 1 external route is the cost shown by the external metric (which is 30 in this case), plus the internal cost to reach the router that advertises the external route.

Then, on another router in the same area, you can see that the default route's cost is 40, because the cost to reach the ASBR is 10. All of the internal routers can see that this is a Type 1 external route, as well as other important attributes such as the administrative distance and the ASBR that originated this route:

```
Router5#show ip route 0.0.0.0
Routing entry for 0.0.0.0/0, supernet
  Known via "ospf 87", distance 110, metric 40, candidate default path
  Tag 55, type extern 1
  Redistributing via ospf 87
  Last update from 172.25.1.5 on Ethernet0, 00:01:24 ago
  Routing Descriptor Blocks:
  * 172.25.1.5, from 172.25.25.1, 00:01:24 ago, via Ethernet0
      Route metric is 40, traffic share count is 1

Router5#
```

With default routes in particular, you sometimes want to ensure that the original ASBR continues to advertise the external route even if it disappears from its routing table. You can do this by adding the keyword *always* to the *default-information* command as follows:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#ip route 0.0.0.0 0.0.0.0  172.25.1.1
Router1(config)#router ospf 55
Router1(config-router)#default-information always metric-type 1
Router1(config-router)#end
Router1#
```

You can also create a default route using a stub area. In this case you can configure your ABR routers to advertise only a simple default route into the area. We discuss stub areas in Recipe 8.10.

8.4.4 See Also

Recipe 8.5; Recipe 8.10

Top

Recipe 8.5 Redistributing Static Routes into OSPF

8.5.1 Problem

You want OSPF to propagate one or more static routes.

8.5.2 Solution

To redistribute static routes into an OSPF process, use the *redistribute static* configuration command:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#ip route 192.168.10.0 255.255.255.0 172.22.1.4
Router1(config)#ip route 172.24.1.0 255.255.255.0 172.22.1.4
Router1(config)#ip route 10.100.1.0 255.255.255.0 172.22.1.4
Router1(config)#router ospf 55
Router1(config-router)#redistribute static
% Only classful networks will be redistributed
Router1(config-router)#end
Router1#
```

8.5.3 Discussion

As the warning message indicates, OSPF will redistribute only classful network routes by default. In the example we included three static routes. Of these routes, only `192.168.10.0/24` is classful. If we look at the routing table on a different router, we can see that the other two routes are not present:

```
Router5#show ip route ospf
O E2 192.168.10.0/24 [110/20] via 172.25.1.5, 00:02:49, Ethernet0
     172.16.0.0/24 is subnetted, 1 subnets
O       172.16.2.0 [110/20] via 172.25.1.5, 00:02:49, Ethernet0
     172.20.0.0/16 is variably subnetted, 3 subnets, 3 masks
O IA    172.20.10.0/24 [110/1582] via 172.25.1.5, 00:02:49, Ethernet0
O IA    172.20.1.0/30 [110/1572] via 172.25.1.5, 00:02:49, Ethernet0
O IA    172.20.100.1/32 [110/1573] via 172.25.1.5, 00:02:49, Ethernet0
     172.22.0.0/24 is subnetted, 1 subnets
O       172.22.1.0 [110/20] via 172.25.1.5, 00:02:49, Ethernet0
     172.25.0.0/16 is variably subnetted, 3 subnets, 2 masks
O       172.25.25.1/32 [110/11] via 172.25.1.5, 00:02:49, Ethernet0
     10.0.0.0/8 is variably subnetted, 2 subnets, 2 masks
O IA    10.2.2.2/32 [110/1573] via 172.25.1.5, 00:02:49, Ethernet0
O IA    10.1.1.0/30 [110/1572] via 172.25.1.5, 00:02:49, Ethernet0
```

```
Router5#
```

You can ensure that all routes are redistributed, classful or not, by including the *subnets* keyword:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#router ospf 55
Router1(config-router)#redistribute static subnets
Router1(config-router)#end
Router1#
```

As you can see, all three static routes are now advertised:

```
Router5#show ip route ospf
O E2 192.168.10.0/24 [110/20] via 172.25.1.5, 00:04:23, Ethernet0
     172.16.0.0/24 is subnetted, 1 subnets
O       172.16.2.0 [110/20] via 172.25.1.5, 00:04:23, Ethernet0
     172.20.0.0/16 is variably subnetted, 3 subnets, 3 masks
O IA    172.20.10.0/24 [110/1582] via 172.25.1.5, 00:04:23, Ethernet0
O IA    172.20.1.0/30 [110/1572] via 172.25.1.5, 00:04:23, Ethernet0
O IA    172.20.100.1/32 [110/1573] via 172.25.1.5, 00:04:23, Ethernet0
     172.22.0.0/24 is subnetted, 1 subnets
O       172.22.1.0 [110/20] via 172.25.1.5, 00:04:23, Ethernet0
     172.25.0.0/16 is variably subnetted, 3 subnets, 2 masks
O       172.25.25.1/32 [110/11] via 172.25.1.5, 00:04:23, Ethernet0
     172.24.0.0/24 is subnetted, 1 subnets
O E2    172.24.1.0 [110/20] via 172.25.1.5, 00:00:24, Ethernet0
     10.0.0.0/8 is variably subnetted, 3 subnets, 3 masks
O IA    10.2.2.2/32 [110/1573] via 172.25.1.5, 00:04:23, Ethernet0
O IA    10.1.1.0/30 [110/1572] via 172.25.1.5, 00:04:23, Ethernet0
O E2    10.100.1.0/24 [110/20] via 172.25.1.5, 00:00:24, Ethernet0
Router5#
```

Another useful thing to notice about this output is the fact that all of these external static routes are marked as type E2, meaning that they are external routes of Type 2. As we discussed in the introduction to this chapter, any time you distribute a foreign route into OSPF, it is always considered external. This allows OSPF to ensure that it doesn't create any loops through an external network when there are multiple connection points.

When OSPF distributes Type 2 external routes, it doesn't add the internal link cost to the net route cost. OSPF always prefers Type 1 to Type 2 external routes because Type 1 routes do include the internal path cost in the metric. If you want to distribute static routes as Type 1 instead of the default Type 2, you need to include the *metric-type* keyword in the *redistribute static* command:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#router ospf 55
Router1(config-router)#redistribute static subnets metric 40 metric-type 1
Router1(config-router)#end
Router1#
```

In this example, we have also set the default metric for these static routes to a value of 40. The next-hop router shows the total cost of the path as 60 because it now includes the internal cost of 20 to reach the ASBR sourcing the external static route:

```
Router5#show ip route 192.168.10.0
Routing entry for 192.168.10.0/24
  Known via "ospf 87", distance 110, metric 60, type extern 1
  Redistributing via ospf 87
  Last update from 172.25.1.5 on Ethernet0, 00:01:20 ago
  Routing Descriptor Blocks:
  * 172.25.1.5, from 172.25.25.1, 00:01:20 ago, via Ethernet0
      Route metric is 60, traffic share count is 1
Router5#
```

Top

Recipe 8.6 Redistributing External Routes into OSPF

8.6.1 Problem

You want OSPF to distribute routes from another routing protocol.

8.6.2 Solution

The *redistribute* configuration command allows you to redistribute routes from another dynamic routing protocol into an OSPF process:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#router ospf 55
Router1(config-router)#redistribute eigrp 11 subnets
Router1(config-router)#end
Router1#
```

8.6.3 Discussion

Redistributing external routes from another routing protocol is similar to redistributing static routes, as we did in Recipe 8.5. In the above example, all of the routes that this router learns through EIGRP process number 11 will be propagated into OSPF as Type 2 external routes. Also, as shown in the following output from the *show ip protocols* command, every route will be redistributed because we also included the *subnets* keyword. If we had not included this keyword, OSPF would only redistribute classful summary routes from EIGRP:

```
Router1#show ip protocols
Routing Protocol is "ospf 55"
  Outgoing update filter list for all interfaces is not set
  Incoming update filter list for all interfaces is not set
  Router ID 172.25.25.1
  It is an area border and autonomous system boundary router
  Redistributing External Routes from,
    static with metric mapped to 40, includes subnets in redistribution
    eigrp 11, includes subnets in redistribution
  Number of areas in this router is 3. 3 normal 0 stub 0 nssa
  Maximum path: 4
  Routing for Networks:
    10.0.0.0 0.255.255.255 area 2
    172.20.0.0 0.0.255.255 area 100
```

```
    0.0.0.0 255.255.255.255 area 0
  Routing Information Sources:
    Gateway         Distance      Last Update
    172.25.1.7           110      00:06:24
    172.25.1.1           110      1d15h
    172.25.1.3           110      00:06:24
  Distance: (default is 110)
Router1#
```

If you prefer, you can redistribute routes from the foreign routing protocol as Type 1 external routes. To do this, you need to specify the *metric-type* keyword in the *redistribute* command:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#router ospf 55
Router1(config-router)#redistribute eigrp 11 subnets metric 35 metric-type 1
Router1(config-router)#end
Router1#
```

You can also do some rather interesting things when redistributing OSPF routes into another protocol. For example, you might choose to only redistribute internal routes to the foreign routing protocol:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#router eigrp 11
Router1(config-router)#redistribute ospf 55 match internal
Router1(config-router)#end
Router1#
```

There are several other options for this *match* keyword. You could just as easily choose to match only external Type 1 routes as follows:

```
Router1(config-router)#redistribute ospf 55 match external type 1
```

You can also combine types to allow you to redistribute both internal and external Type 1 routes, but not external Type 2:

```
Router1(config-router)#redistribute ospf 55 match internal match external type 1
```

This *match* option on the *redistribute* command is much easier than configuring a route map. But, if you require still greater control and flexibility, route maps are the best choice. This is particularly true if you want to handle routing tags in a special way. For a discussion of using route maps while redistributing routes between protocols, please refer to Recipe 8.13 on OSPF route tagging.

8.6.4 See Also

Recipe 8.5; Recipe 8.13

Top

Recipe 8.7 Manipulating DR Selection

8.7.1 Problem

You want to manipulate the Designated Router (DR) selection process on a particular subnet.

8.7.2 Solution

The *ip ospf priority* configuration command allows you to weight the DR selection process on a network segment. The following configuration examples are for three different routers that all share the same Ethernet segment. Router5 has the highest OSPF priority, so it will become the DR. Router1 has the second highest priority because we want it to be the Backup Designated Router (BDR).

Router1 is connected to this network segment through a VLAN trunk:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#interface FastEthernet0/0.1
Router1(config-subif)#ip ospf priority 2
Router1(config-subif)#end
Router1#
```

We will configure Router3 with a priority of 0. The default priority is 1. A router with priority 0 will never become the DR or BDR:

```
Router3#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router3(config)#interface FastEthernet0/0.1
Router3(config-subif)#ip ospf priority 0
Router3(config-subif)#end
Router3#
```

Router5 has the highest priority, so it will become the DR for the segment:

```
Router5#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router5(config)#interface Ethernet 0
Router5(config-if)#ip ospf priority 10
Router5(config-if)#end
Router5#
```

8.7.3 Discussion

There are several reasons for rigging the DR election process, as we have done in this recipe. The most common reason is simply to ensure that the router closest to the network core is responsible for distributing routing information. This is actually a somewhat aesthetic requirement, because all of the routers in an area see all of the LSAs for that area. Nobody's routing table is more accurate than anybody else's, but making the router closest to the core be the DR can result in faster convergence for some network configurations.

There are two times when it is critical to force a particular router to become the DR. The first is when you are using MOSPF to handle multicast routing. MOSPF uses the same DR as regular OSPF. So, if you have a mix of MOSPF and regular OSPF on the same segment, it is critical that an MOSPF router be the DR, or no multicast routes will be distributed. Since Cisco routers do not support MOSPF, this means that you must set the priority to 0 for all Cisco routers on such a segment.

The second place where DR selection is critical is in Non-Broadcast Multiple Access (NBMA) networks. A typical example of this would be a Frame Relay WAN that uses multipoint subinterfaces, as described in Recipe 10.4. In this case, all of the routers are members of the same subnet, but only the central hub router can talk directly to the branch devices. A branch router should never act as DR because it can't talk directly to any of the other branches. The central router is the only device that can be the DR, or the routing updates will not work.

If you don't adjust the priorities to help force a particular winner to the DR election, the DR will be the router with the highest Router ID (RID) value. Please see Recipe 8.8 for a discussion of RID values.

It is important to note that setting a higher priority can help to rig the DR election process, but it doesn't guarantee that another lower priority router device won't become DR if it happens to be there first. If a higher priority router comes up on a segment that already has a DR, it will not preempt either the DR or the BDR. If the higher priority router isn't available when a lower priority router joins the segment, then the lower priority router will become DR. Once a router is DR, it will remain DR until you either manually reset the neighbor relationships or until there is a network failure on the segment that forces the change.

The exception to this rule is when you configure a router with a priority of 0. In this case, the router will never become DR, even if it is the only router on the segment.

You can see the state of all of the neighboring routers on a segment with the show *ip ospf neighbor* command:

```
Router5#show ip ospf neighbor Ethernet0

Neighbor ID     Pri   State           Dead Time   Address         Interface
Router1           2   FULL/BDR        00:00:31    172.25.1.5      Ethernet0
Router3           0   FULL/DROTHER    00:00:31    172.25.1.3      Ethernet0
Router4           1   FULL/DROTHER    00:00:39    172.25.1.1      Ethernet0
Router5#
```

In this output, we have asked the router to only show the neighbors on the `Ethernet0` interface. You can see that Router1 is the BDR, and the other two routers on the segment have a state of "DROTHER." This means that they are neither DR nor BDR, but that they are neighbors. Notice that none of the routers listed is the DR. This is because the router we typed this command on was the DR itself.

You can verify that this router is the DR, and that it has a priority of 10 with the *show ip ospf interface* command:

```
Router5#show ip ospf interface
Ethernet0 is up, line protocol is up
  Internet Address 172.25.1.7/24, Area 0
  Process ID 87, Router ID 172.25.1.7, Network Type BROADCAST, Cost: 10
  Transmit Delay is 1 sec,  State DR, Priority 10
  Designated Router (ID) 172.25.25.6, Interface address 172.25.1.7
  Backup Designated router (ID) Router1, Interface address 172.25.1.3
  Timer intervals configured, Hello 10, Dead 40, Wait 40, Retransmit 5
    Hello due in 00:00:03
  Neighbor Count is 3, Adjacent neighbor count is 3
    Adjacent with neighbor Router3
    Adjacent with neighbor Router1  (Backup Designated Router)
    Adjacent with neighbor Router4
  Suppress hello for 0 neighbor(s)
Router5#
```

In the following example, we have increased the priority of Router4 to 10 as well. However, as you can see, not only does it not preempt the DR, it doesn't even preempt the BDR:

```
Router5#show ip ospf neighbor

Neighbor ID     Pri   State           Dead Time   Address         Interface
Router4          10   FULL/DROTHER     00:00:30    172.25.1.5      Ethernet0
Router1           2   FULL/BDR         00:00:38    172.25.1.3      Ethernet0
Router3           0   FULL/DROTHER    00:00:30    172.25.1.1      Ethernet0
Router5#
```

Because higher priority routers will not preempt existing DR and BDR routers, if there are routers that should not become DR for any reason, you should be careful to set their priorities to 0. Otherwise, you may find that the DR is simply the router that has been active for the longest time, instead of the one that you actually wanted.

8.7.4 See Also

Recipe 8.8; Recipe 10.4; Recipe 23.7

Top

Recipe 8.8 Setting the OSPF RID

8.8.1 Problem

You want to set the OSPF Router ID (RID) of a particular router.

8.8.2 Solution

There are several ways to set the OSPF RID. The easiest is to create and configure a loopback interface:

```
Router5#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router5(config)#interface Loopback0
Router5(config-if)#ip address 172.25.25.6 255.255.255.255
Router5(config-if)#end
Router5#
```

If you don't want to use a Loopback interface, you can still force the router ID to use a particular IP address with the *router-id* configuration command:

```
Router5#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router5(config)#router ospf 87
Router5(config-router)#router-id 172.25.1.7
Router5(config-router)#end
Router5#
```

8.8.3 Discussion

If you don't use either of these methods, the router will select the highest IP address from its interfaces and use this as the OSPF RID. The trouble with doing this is that you might add a new IP address to the router at some point. If this new address is higher than the previous RID, the router will change its RID the next time OSPF restarts. This could have strange consequences because, if the interface priorities are the same, OSPF uses the highest RID to select the DR. Please refer to Recipe 8.7 for more information on DR selection.

We recommend using the loopback interface method. Loopback interfaces also ensure that there is a single unique IP address for every router in the network, which is extremely useful for network management. Further, it is common to configure your loopback addresses in DNS, but not to necessarily include all of your interfaces. This is useful if you enable domain name lookups on your

router, as discussed in Recipe 8.16.

In IOS Version 12.0, Cisco introduced a new way to select the RID using the *router-id* command. This command allows you to set the RID to any IP address. You can even set the RID to be an address that is not configured on any of the router's interfaces or even in any routing tables. However, this is not a very wise thing to do because it makes troubleshooting much more difficult.

In some cases, you might have both a router-id and a loopback address set. The rule is that OSPF will use the *router-id* command first, if one exists. If there is no router-id command, then it uses the highest IP address on any of the loopback interfaces. Bear in mind that you can configure as many loopback interfaces as you like (although this is somewhat unusual in production networks, there are special situations where additional loopback interfaces can be useful). Finally, if there is no router-id command and no loopback interface, the OSPF process will use the highest IP address on the router for the RID.

You can see what the RID for your router is with the following command:

```
Router5#show ip ospf
 Routing Process "ospf 87" with ID 172.25.1.7
 Supports only single TOS(TOS0) routes
 SPF schedule delay 5 secs, Hold time between two SPFs 10 secs
 Minimum LSA interval 5 secs. Minimum LSA arrival 1 secs
 Number of external LSA 5. Checksum Sum 0x28868
 Number of DCbitless external LSA 0
 Number of DoNotAge external LSA 0
 Number of areas in this router is 1. 1 normal 0 stub 0 nssa
    Area BACKBONE(0)
        Number of interfaces in this area is 2
        Area has no authentication
        SPF algorithm executed 47 times
        Area ranges are
        Number of LSA 36. Checksum Sum 0xEEAA1
        Number of DCbitless LSA 9
        Number of indication LSA 0
        Number of DoNotAge LSA 0
Router5#
```

The router will continue to use the same RID address, even if you subsequently add a router-id command or a loopback interface. To force OSPF to update the RID, you can either reload the router or restart the OSPF process using the *clear ip ospf process* command:

```
Router5#clear ip ospf process
Reset ALL OSPF processes? [no]: yes
Router5#
```

8.8.4 See Also

Recipe 8.7; Recipe 8.16

Top

Recipe 8.9 Enabling OSPF Authentication

8.9.1 Problem

You want to authenticate your OSPF neighbor relationships to ensure that no unauthorized equipment is allowed to affect routing.

8.9.2 Solution

To enable OSPF MD5 authentication, you need to define the encryption key, which is essentially just a password on an interface. You must also enable authentication for the entire area. For the first router, you could do this as follows:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#interface Serial0/1
Router1(config-if)#ip ospf message-digest-key 1 md5 oreilly
Router1(config-if)#exit
Router1(config)#router ospf 55
Router1(config-router)#area 2 authentication message-digest
Router1(config-router)#end
Router1#
```

Similarly, you must enable OSPF authentication on other routers in the area, making sure that the authentication keys match on all interfaces that share the same network segment:

```
Router2#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router2(config)#interface Serial0/0
Router2(config-if)#ip ospf message-digest-key 1 md5  oreilly
Router2(config-if)#exit
Router2(config)#router ospf 12
Router2(config-router)#area 2 authentication message-digest
Router2(config-router)#end
Router2#
```

8.9.3 Discussion

RFC 2328, which defines OSPF Version 2, includes three different types of authentication for OSPF: null authentication, simple password authentication, and cryptographic authentication. Null authentication simply means that there is no authentication, which is the default on Cisco routers. In the

simple password method of authentication, passwords are exchanged in clear-text on the network. Even the RFC that specifies this method points out that it is easily compromised. Anybody who wants to deliberately corrupt your routing tables needs to have direct access to your network to do so anyway. Having that access means that it is relatively easy to capture these passwords. We recommend that you use the cryptographic authentication method if you require authentication with OSPF.

The cryptographic method uses the open standard Message Digest Type 5 (MD5) encryption standard. MD5 is a one-way irreversible cipher. Two devices exchange only the MD5 encrypted versions of the password. Both devices know the same password, and each router is able to verify that the encrypted password that it receives is correct by using the same algorithm to encrypt the password that it already knows. To make sure that nobody can just intercept and use the encrypted version of the password directly, a time value that the receiving router also knows is added to the password before encrypting. Anybody else listening on the network is only able to see the encrypted version of the password, but they cannot deduce the original password.

Unfortunately, the RFC is not completely clear on how this time value should be added to the original pass phrase, nor does it mandate MD5 encryption. So there is a good chance that cryptographic authentication will not work well between routers from different vendors.

If you use authentication in an OSPF area, you must configure all of the routers in the area to support authentication. Every interface on a router doesn't have to be configured with authentication. But if you require authentication in any part of an area, you must include authentication support throughout the area. In the above example, this is done for Area 2 with the following command:

```
Router2(config-router)#area 2 authentication message-digest
```

The *show ip ospf interface* command shows that we have configured authentication on this interface:

```
Router2#show ip ospf interface Serial0/0
Serial0/0 is up, line protocol is up
  Internet Address 10.1.1.1/30, Area 2
  Process ID 12, Router ID 192.168.30.1, Network Type POINT_TO_POINT, Cost: 130
  Transmit Delay is 1 sec, State POINT_TO_POINT,
  Timer intervals configured, Hello 10, Dead 40, Wait 40, Retransmit 5
    Hello due in 00:00:06
  Index 1/1, flood queue length 0
  Next 0x0(0)/0x0(0)
  Last flood scan length is 1, maximum is 1
  Last flood scan time is 0 msec, maximum is 0 msec
  Neighbor Count is 1, Adjacent neighbor count is 1
    Adjacent with neighbor 172.25.25.1
  Suppress hello for 0 neighbor(s)
  Message digest authentication enabled
    Youngest key id is 1
Router2#
```

Note that we are using "Message digest authentication," (meaning MD5) and that key number 1 is

currently active.

You can use a different key on each of a router's interfaces, or a single password throughout the entire network. All that matters is that all of the routers on a single network segment use the same OSPF key for the interfaces that share this segment. The problem with using too many different keys is that it can become rather difficult to manage.

You can also configure several keys on a single interface. We recommend using this as a transition method while changing keys, and the old keys should be removed quickly to prevent anybody from gaining access using an old key:

```
Router2#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router2(config)#interface Serial0/0
Router2(config-if)#ip ospf message-digest-key 1 md5 oreilly
Router2(config-if)#ip ospf message-digest-key 2 md5 cookbook
Router2(config-if)#end
Router2#
```

In this case we have defined two keys, which have key numbers 1 and 2, respectively:

```
Router2#show ip ospf interface Serial0/0
Serial0/0 is up, line protocol is up
  Internet Address 10.1.1.1/30, Area 2
  Process ID 12, Router ID 192.168.30.1, Network Type POINT_TO_POINT, Cost: 130
  Transmit Delay is 1 sec, State POINT_TO_POINT,
  Timer intervals configured, Hello 10, Dead 40, Wait 40, Retransmit 5
    Hello due in 00:00:03
  Index 1/1, flood queue length 0
  Next 0x0(0)/0x0(0)
  Last flood scan length is 1, maximum is 1
  Last flood scan time is 0 msec, maximum is 0 msec
  Neighbor Count is 1, Adjacent neighbor count is 1
    Adjacent with neighbor 172.25.25.1
  Suppress hello for 0 neighbor(s)
  Message digest authentication enabled
    Youngest key id is 2
    Rollover in progress, 1 neighbor(s) using the old key(s):
      key id 1
Router2#
```

This display indicates that key number 2 is the newest, and that one neighbor is still using the old key. This command is useful when you want to see if it is safe to remove the old key yet.

Looking at the router's configuration file, you can see that these keys are stored in plain-text by default:

```
interface Serial0/0
 ip address 10.1.1.1 255.255.255.252
 ip ospf message-digest-key 1 md5 oreilly
 ip ospf message-digest-key 2 md5 cookbook
```

If you define the password encryption service on the router, it will store these keys using the weak Cisco type 7 encryption method:

```
Router2#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router2(config)#service password-encryption
Router2(config)#end
```

As we discussed in Chapter 2, this causes the router to store passwords in an encrypted form when you view the configuration file. However, this encryption method can easily be broken if somebody gains access to the router. It is still useful, though, to prevent somebody from getting the passwords by looking over your shoulder.

If you want to use authentication, but the neighboring devices don't support MD5, use clear-text authentication:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#interface Serial0/1
Router1(config-if)#ip ospf authentication-key oreilly
Router1(config-if)#exit
Router1(config)#router ospf 55
Router1(config-router)#area 2 authentication
Router1(config-router)#end
Router1#
```

As with MD5 authentication, if you configure clear-text authentication on an interface, you must configure the same authentication method and the same key on all other routers that share this segment:

```
Router2#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router2(config)#interface Serial0/0
Router2(config-if)#ip ospf authentication-key oreilly
Router2(config-if)#exit
Router2(config)#router ospf 12
Router2(config-router)#area 2 authentication
Router2(config-router)#end
Router2#
```

The output of the *show ip ospf interface* command now indicates the alternative authentication method:

```
Router2#show ip ospf interface Serial0/0
Serial0/0 is up, line protocol is up
  Internet Address 10.1.1.1/30, Area 2
  Process ID 12, Router ID 192.168.30.1, Network Type POINT_TO_POINT, Cost: 130
  Transmit Delay is 1 sec, State POINT_TO_POINT,
  Timer intervals configured, Hello 10, Dead 40, Wait 40, Retransmit 5
    Hello due in 00:00:07
  Index 1/1, flood queue length 0
  Next 0x0(0)/0x0(0)
```

```
  Last flood scan length is 1, maximum is 1
  Last flood scan time is 0 msec, maximum is 0 msec
  Neighbor Count is 1, Adjacent neighbor count is 1
    Adjacent with neighbor 172.25.25.1
  Suppress hello for 0 neighbor(s)
  Simple password authentication enabled
Router2#
```

8.9.4 See Also

Chapter 2

Top

Recipe 8.10 Selecting the Appropriate Area Types

8.10.1 Problem

You want to limit the number of routes and entries in the Link State database to conserve router resources and ensure good convergence properties.

8.10.2 Solution

In the introduction to this chapter, we talked about the various types of OSPF areas. You can configure these different types areas using the appropriate keywords on the *area* command.

For a stubby area, use the *stub* keyword:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#router ospf 55
Router1(config-router)#area 100 stub
Router1(config-router)#end
Router1#
```

To configure a totally stubby area, combine the *stub* and *no-summary* keywords on the ABR router:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#router ospf 55
Router1(config-router)#area 100 stub no-summary
Router1(config-router)#end
Router1#
```

The other routers in a totally stubby area need only be configured using the *stub* keyword, as in the previous example.

For not so stubby areas (NSSA), you need to specify the *nssa* keyword. In this case we have also included the *default-information-originate* option so that the router can summarize external routes to a single default route:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#router ospf 55
Router1(config-router)#area 100 nssa default-information-originate
Router1(config-router)#end
```

```
Router1#
```

In the introduction, we also discussed an interesting variant called the totally stubby not so stubby area. You can configure this as follows on the ABR:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#router ospf 55
Router1(config-router)#area 100 nssa no-summary
Router1(config-router)#end
Router1#
```

Once again, you can simply configure the other routers in this area with the *nssa* keyword.

8.10.3 Discussion

In all of the configuration examples, we showed the configuration for just one router. It is important to remember that the routers in an area have to agree on the area type. However, it is mostly the ABR that cares about the area type, because it has to decide what kinds of information to forward from other parts of the network. Every router in a stub area must be configured as stub, although only the ABR needs to worry about the difference between stub and totally stub. Every router in an NSSA area must be configured to support NSSA Type 7 LSA messages. But, once again, only the ABR cares about the difference between NSSA and totally stub NSSA.

Figure 8-1 shows the network diagram for all of these examples. Note that the diagram includes detailed information for everything inside of Area 100, but only summary information for everything in other areas and other networks. This is how OSPF views the world.

Figure 8-1. Example network for area type examples

If you have any ASBR devices in this area that inject routes from other Autonomous Systems, then they also need to use the area type information. The other routers are only really concerned with forwarding the LSAs around the area.

When you look at the routing table, you can see that there are several kinds of routes:

```
Router3#sh ip route ospf
O E1 192.168.10.0/24 [110/3611] via 172.20.1.1, 00:00:20, Serial0.1
     172.16.0.0/24 is subnetted, 1 subnets
O IA    172.16.2.0 [110/3581] via 172.20.1.1, 00:00:20, Serial0.1
     172.20.0.0/16 is variably subnetted, 5 subnets, 3 masks
O       172.20.220.1/32 [110/11] via 172.20.10.2, 00:00:20, Ethernet0
O       172.20.200.1/32 [110/11] via 172.20.10.2, 00:00:20, Ethernet0
     172.22.0.0/24 is subnetted, 1 subnets
O IA    172.22.1.0 [110/3581] via 172.20.1.1, 00:00:20, Serial0.1
     172.25.0.0/16 is variably subnetted, 3 subnets, 2 masks
O IA    172.25.25.6/32 [110/3582] via 172.20.1.1, 00:00:20, Serial0.1
O IA    172.25.25.1/32 [110/3572] via 172.20.1.1, 00:00:20, Serial0.1
O IA    172.25.1.0/24 [110/3581] via 172.20.1.1, 00:00:20, Serial0.1
     172.24.0.0/24 is subnetted, 1 subnets
O E1    172.24.1.0 [110/3611] via 172.20.1.1, 00:00:20, Serial0.1
     10.0.0.0/8 is variably subnetted, 4 subnets, 3 masks
O IA    10.2.2.2/32 [110/5134] via 172.20.1.1, 00:00:20, Serial0.1
O E1    10.2.2.0/30 [110/3606] via 172.20.1.1, 00:00:20, Serial0.1
O IA    10.1.1.0/30 [110/5133] via 172.20.1.1, 00:00:20, Serial0.1
O E1    10.100.1.0/24 [110/3611] via 172.20.1.1, 00:00:20, Serial0.1
O E2 192.168.50.0/24 [110/20] via 172.20.1.1, 00:00:20, Serial0.1
Router3#
```

This output shows only routes that were learned via OSPF, which is indicated by the `O` at the start of each line. In this routing table there are several external routes. Most of these are Type 1 externals, which are labeled `E1`. There is also one Type 2 external route, which is labeled `E2` in the output. Some of the other routes have `IA` beside them, which indicates that they are inter-area routes, meaning that they did not originate in this area. The remaining routes, such as `172.20.220.1/32`, represent networks in this area.

This routing table shows no summarization. It's also useful to look at the OSPF database on this router to see how different routes are categorized:

```
Router3#show ip ospf database

            OSPF Router with ID (172.25.25.2) (Process ID 44)

                Router Link States (Area 100)

Link ID         ADV Router      Age         Seq#       Checksum Link count
172.20.220.1    172.20.220.1    47          0x80000004 0xB352   3
172.25.25.1     172.25.25.1     89          0x80000067 0xE771   2
172.25.25.2     172.25.25.2     47          0x80000065 0x4C66   4
```

```
                Net Link States (Area 100)

Link ID         ADV Router      Age         Seq#       Checksum
172.20.10.1     172.25.25.2     42          0x80000002 0xF11B

                Summary Net Link States (Area 100)

Link ID         ADV Router      Age         Seq#       Checksum
10.1.1.0        172.25.25.1     173         0x80000002 0x86AC
10.2.2.2        172.25.25.1     173         0x80000002 0x77B3
172.16.2.0      172.25.25.1     173         0x80000002 0xBFD3
172.22.1.0      172.25.25.1     173         0x80000002 0x820C
172.25.1.0      172.25.25.1     173         0x80000002 0x5E2D
172.25.25.1     172.25.25.1     173         0x80000002 0xF08A
172.25.25.6     172.25.25.1     173         0x80000002 0x2349

                Summary ASB Link States (Area 100)

Link ID         ADV Router      Age         Seq#       Checksum
172.25.1.7      172.25.25.1     173         0x80000001 0xC78
172.25.25.1     172.25.25.1     173         0x80000001 0xBCDF

                Type-5 AS External Link States

Link ID         ADV Router      Age         Seq#       Checksum Tag
10.2.2.0        172.25.25.1     1138        0x8000000A 0x9588   0
10.100.1.0      172.25.25.1     1138        0x80000009 0x4A6B   0
172.24.1.0      172.25.25.1     1138        0x80000009 0x9BC3   0
192.168.10.0    172.25.25.1     1138        0x80000009 0x6C45   0
192.168.50.0    172.25.1.7      428         0x80000002 0xFF36   0
Router3#
```

This represents a relatively small OSPF network, but it's a useful example because there are some instances of each type of route.

When we configure this area to be a stubby area, the ABR prevents external routes from being propagated into the area and replaces them with a default route. ASBRs are not permitted in stubby areas:

```
Router3#show ip route ospf
     172.16.0.0/24 is subnetted, 1 subnets
O IA    172.16.2.0 [110/3581] via 172.20.1.1, 00:00:07, Serial0.1
     172.20.0.0/16 is variably subnetted, 5 subnets, 3 masks
O       172.20.220.1/32 [110/11] via 172.20.10.2, 00:00:20, Ethernet0
O       172.20.200.1/32 [110/11] via 172.20.10.2, 00:00:20, Ethernet0
     172.22.0.0/24 is subnetted, 1 subnets
O IA    172.22.1.0 [110/3581] via 172.20.1.1, 00:00:07, Serial0.1
     172.25.0.0/16 is variably subnetted, 3 subnets, 2 masks
O IA    172.25.25.6/32 [110/3582] via 172.20.1.1, 00:00:07, Serial0.1
O IA    172.25.25.1/32 [110/3572] via 172.20.1.1, 00:00:07, Serial0.1
```

```
O IA     172.25.1.0/24 [110/3581] via 172.20.1.1, 00:00:07, Serial0.1
      10.0.0.0/8 is variably subnetted, 2 subnets, 2 masks
O IA     10.2.2.2/32 [110/5134] via 172.20.1.1, 00:00:07, Serial0.1
O IA     10.1.1.0/30 [110/5133] via 172.20.1.1, 00:00:07, Serial0.1
O*IA 0.0.0.0/0 [110/3572] via 172.20.1.1, 00:00:07, Serial0.1
Router3#
```

As you can see, all of the external routes are gone, but the inter-area routes remain. Looking at the OSPF database, you can see that there is considerably less information for the router to keep track of:

```
Router3#show ip ospf database

            OSPF Router with ID (172.25.25.2) (Process ID 44)

                Router Link States (Area 100)

Link ID         ADV Router      Age         Seq#       Checksum Link count
172.20.220.1    172.20.220.1    22          0x80000006 0xCD38   3
172.25.25.1     172.25.25.1     86          0x80000069 0xFB5F   2
172.25.25.2     172.25.25.2     22          0x80000068 0x644D   4

                Net Link States (Area 100)

Link ID         ADV Router      Age         Seq#       Checksum
172.20.10.1     172.25.25.2     17          0x80000003 0xEFF

                Summary Net Link States (Area 100)

Link ID         ADV Router      Age         Seq#       Checksum
0.0.0.0         172.25.25.1     92          0x80000001 0x213D
10.1.1.0        172.25.25.1     92          0x80000003 0xA291
10.2.2.2        172.25.25.1     92          0x80000003 0x9398
172.16.2.0      172.25.25.1     92          0x80000003 0xDBB8
172.22.1.0      172.25.25.1     92          0x80000003 0x9EF0
172.25.1.0      172.25.25.1     92          0x80000003 0x7A12
172.25.25.1     172.25.25.1     92          0x80000003 0xD6F
172.25.25.6     172.25.25.1     92          0x80000003 0x3F2E
Router3#
```

Totally stubby areas, like ordinary stub areas, also prevent external routes. But the ABRs for totally stubby areas also prevent inter-area routes from being propagated into the area, replacing them with a single default route instead. The default route is the only summary route allowed. ASBRs are not permitted in stubby or totally stubby areas:

```
Router3#show ip route ospf
      172.20.0.0/16 is variably subnetted, 5 subnets, 3 masks
O        172.20.220.1/32 [110/11] via 172.20.10.2, 00:00:15, Ethernet0
O        172.20.200.1/32 [110/11] via 172.20.10.2, 00:00:15, Ethernet0
O*IA 0.0.0.0/0 [110/3572] via 172.20.1.1, 00:00:15, Serial0.1
Router3#
```

Clearly, the totally stubby area has radically reduced the size of the routing table, as well as the OSPF database:

```
Router3#show ip ospf database

            OSPF Router with ID (172.25.25.2) (Process ID 44)

                Router Link States (Area 100)

Link ID         ADV Router      Age         Seq#       Checksum Link count
172.20.220.1    172.20.220.1    104         0x80000006 0xCD38   3
172.25.25.1     172.25.25.1     22          0x8000006B 0xF761   2
172.25.25.2     172.25.25.2     104         0x80000068 0x644D   4

                Net Link States (Area 100)

Link ID         ADV Router      Age         Seq#       Checksum
172.20.10.1     172.25.25.2     99          0x80000003 0xEFF

                Summary Net Link States (Area 100)

Link ID         ADV Router      Age         Seq#       Checksum
0.0.0.0         172.25.25.1     23          0x80000002 0x1F3E
Router3#
```

There are two main differences between stubby and NSSA areas. The first is that the NSSA prevents the ABR from propagating external routes throughout the area, but does not replace them with a single default route. The second is that the NSSA may contain ASBRs, which use Type 7 LSAs to carry information about external routes. These Type 7 LSAs are flooded throughout the NSSA area. When they reach the ABR, they are translated into Type 5 LSAs and forwarded to the rest of the OSPF network:

```
Router3#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router3(config)#ip route 192.168.88.0 255.255.255.0 172.20.10.2
Router3(config)#router ospf 44
Router3(config-router)#redistribute static subnet
Router3(config-router)#area 100 nssa default-information-originate
Router3(config-router)#end
Router3#
```

With the *default-information-originate* option, the ABR will forward a default route to summarize all external routes that originate outside of the area:

```
Router3#show ip route ospf
     172.16.0.0/24 is subnetted, 1 subnets
O IA    172.16.2.0 [110/3581] via 172.20.1.1, 00:07:43, Serial0.1
     172.20.0.0/16 is variably subnetted, 5 subnets, 3 masks
O       172.20.220.1/32 [110/11] via 172.20.10.2, 00:07:43, Ethernet0
O       172.20.200.1/32 [110/11] via 172.20.10.2, 00:07:43, Ethernet0
```

```
      172.22.0.0/24 is subnetted, 1 subnets
O IA     172.22.1.0 [110/3581] via 172.20.1.1, 00:07:43, Serial0.1
      172.25.0.0/16 is variably subnetted, 3 subnets, 2 masks
O IA     172.25.25.6/32 [110/3582] via 172.20.1.1, 00:07:43, Serial0.1
O IA     172.25.25.1/32 [110/3572] via 172.20.1.1, 00:07:43, Serial0.1
O IA     172.25.1.0/24 [110/3581] via 172.20.1.1, 00:07:43, Serial0.1
      10.0.0.0/8 is variably subnetted, 2 subnets, 2 masks
O IA     10.2.2.2/32 [110/5134] via 172.20.1.1, 00:07:43, Serial0.1
O IA     10.1.1.0/30 [110/5133] via 172.20.1.1, 00:07:43, Serial0.1
O*N2 0.0.0.0/0 [110/1] via 172.20.1.1, 00:07:43, Serial0.1
Router3#
```

The OSPF database for an NSSA area includes information about Type 7 LSAs:

```
Router3#show ip ospf database

            OSPF Router with ID (172.25.25.2) (Process ID 44)

                Router Link States (Area 100)

Link ID         ADV Router      Age         Seq#       Checksum Link count
172.20.220.1    172.20.220.1    973         0x80000008 0x51AA   3
172.25.25.1     172.25.25.1     502         0x80000072 0x77D0   2
172.25.25.2     172.25.25.2     968         0x8000006E 0xE5BB   4

                Net Link States (Area 100)

Link ID         ADV Router      Age         Seq#       Checksum
172.20.10.1     172.25.25.2     967         0x80000004 0x9371

                Summary Net Link States (Area 100)

Link ID         ADV Router      Age         Seq#       Checksum
10.1.1.0        172.25.25.1     1124        0x80000003 0x2A02
10.2.2.2        172.25.25.1     1124        0x80000003 0x1B09
172.16.2.0      172.25.25.1     1124        0x80000003 0x6329
172.22.1.0      172.25.25.1     1124        0x80000003 0x2661
172.25.1.0      172.25.25.1     1124        0x80000003 0x282
172.25.25.1     172.25.25.1     1124        0x80000003 0x94DF
172.25.25.6     172.25.25.1     1124        0x80000003 0xC69E

                Type-7 AS External Link States (Area 100)

Link ID         ADV Router      Age         Seq#       Checksum Tag
0.0.0.0         172.25.25.1     508         0x80000001 0xF31B   0
10.2.2.0        172.25.25.1     1123        0x80000001 0x2FE7   0
10.100.1.0      172.25.25.1     1123        0x80000001 0xE90B   0
172.24.1.0      172.25.25.1     1123        0x80000001 0x3B63   0
192.168.10.0    172.25.25.1     1123        0x80000001 0xCE4    0
192.168.88.0    172.25.25.2     974         0x80000001 0x3AEC   0
Router3#
```

It is also interesting to look at the routing table entry for one of these Type 7 routes on another router within the same area:

```
Router1#show ip route 192.168.88.0
Routing entry for 192.168.88.0/24
  Known via "ospf 55", metric 20, type NSSA extern 2 , forward metric 1572
  Last update from 172.20.1.2 on Serial0/0.2, 00:08:56 ago
  Routing Descriptor Blocks:
  * 172.20.1.2, from 172.25.25.2, 00:08:56 ago, via Serial0/0.2
      Route metric is 20, traffic share count is 1
Router1#
```

The aptly named totally stubby not so stubby area is similar to an NSSA area, but it also acts like a totally stubby area by preventing the ABR from advertising inter-area routes and replaces them with a single summary route, the default route. The ABR for a totally stubby NSSA will create a default route by default:

```
Router3#show ip route ospf
     172.20.0.0/16 is variably subnetted, 5 subnets, 3 masks
O       172.20.220.1/32 [110/11] via 172.20.10.2, 00:00:47, Ethernet0
O       172.20.200.1/32 [110/11] via 172.20.10.2, 00:00:47, Ethernet0
O*IA 0.0.0.0/0 [110/3572] via 172.20.1.1, 00:00:47, Serial0.1
Router3#
```

Despite the confusing name, this is an extremely useful type of area. In many large OSPF networks, most of the routes in the routing table are inter-area routes. But you can also put an ASBR in this type of area and use it as a transit area to connect to external networks:

```
Router3#show ip ospf database

            OSPF Router with ID (172.25.25.2) (Process ID 44)

                Router Link States (Area 100)

Link ID         ADV Router      Age         Seq#       Checksum Link count
172.20.220.1    172.20.220.1    1209        0x80000008 0x51AA   3
172.25.25.1     172.25.25.1     91          0x80000074 0x73D2   2
172.25.25.2     172.25.25.2     1204        0x8000006E 0xE5BB   4

                Net Link States (Area 100)

Link ID         ADV Router      Age         Seq#       Checksum
172.20.10.1     172.25.25.2     1203        0x80000004 0x9371

                Summary Net Link States (Area 100)

Link ID         ADV Router      Age         Seq#       Checksum
0.0.0.0         172.25.25.1     92          0x80000001 0xA8AD

                Type-7 AS External Link States (Area 100)
```

```
Link ID         ADV Router      Age         Seq#       Checksum Tag
10.2.2.0        172.25.25.1     88          0x80000002 0x2DE8   0
10.100.1.0      172.25.25.1     82          0x80000003 0xE50D   0
172.24.1.0      172.25.25.1     88          0x80000002 0x3964   0
192.168.10.0    172.25.25.1     88          0x80000002 0xAE5    0
192.168.88.0    172.25.25.2     86          0x80000002 0x38ED   0
Router3#
```

Like an NSSA area, the ABR translates Type 7 to Type 5 LSAs:

```
Router1#show ip ospf
 Routing Process "ospf 55" with ID 172.25.25.1
  <lines removed for brevity>
    Area 100
        Number of interfaces in this area is 1
        It is a NSSA area
        Perform type-7/type-5 LSA translation
        Area has no authentication
        SPF algorithm executed 75 times
        Area ranges are
        Number of LSA 13. Checksum Sum 0x6B01B
        Number of opaque link LSA 0. Checksum Sum 0x0
        Number of DCbitless LSA 0
        Number of indication LSA 0
        Number of DoNotAge LSA 0
        Flood list length 0
Router1#
```

8.10.4 See Also

IP Routing, by Ravi Malhotra(O'Reilly)

Top

Recipe 8.11 Summarizing Routes in OSPF

8.11.1 Problem

You want to reduce the size of your routing tables without losing any connectivity within your network.

8.11.2 Solution

Using the *area x range* configuration command on your ABRs allows you to summarize routes between OSPF areas.

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#router ospf 55
Router1(config-router)#area 100 range 172.20.0.0 255.255.0.0
Router1(config-router)#area 0 range 172.25.0.0 255.255.0.0
Router1(config-router)#area 2 range 10.0.0.0 255.0.0.0
Router1(config-router)#end
Router1#
```

8.11.3 Discussion

The easiest way to see the effect of summarization is to look at the routing table before and after it is enabled. Here is a sample routing table before summarization. The ranges that we will be summarizing are `172.20.0.0/16` and `172.25.0.0/16`. We have highlighted the lowest metric in each of these ranges:

```
Router2#show ip route ospf
     172.16.0.0/24 is subnetted, 1 subnets
O IA    172.16.2.0 [110/140] via 10.1.1.2, 00:05:06, Serial0/0
     172.20.0.0/16 is variably subnetted, 3 subnets, 3 masks
O IA    172.20.10.0/24 [110/1702] via 10.1.1.2, 00:05:06, Serial0/0
O IA    172.20.1.0/30 [110/ 1692] via 10.1.1.2, 00:05:06, Serial0/0
O IA    172.20.100.1/32 [110/1693] via 10.1.1.2, 00:05:06, Serial0/0
     172.25.0.0/16 is variably subnetted, 3 subnets, 2 masks
O IA    172.25.25.6/32 [110/141] via 10.1.1.2, 00:05:06, Serial0/0
O IA    172.25.25.1/32 [110/131] via 10.1.1.2, 00:05:06, Serial0/0
O IA    172.25.1.0/24 [110/140] via 10.1.1.2, 00:00:25, Serial0/0
Router2#
```

After enabling summarization, you can see that all of the individual routes in each of these ranges has

been replaced with a single route for the entire range. Also note that the cost for the summary is equal to the lowest cost of the individual routes that this summary replaces. This is an extremely important point because it means that a summary route will be only as stable as the lowest cost route being replaced. The summary could replace a hundred routes that are all rock stable, but if the one with the lowest cost happens to bounce up and down frequently, the cost of the summary will have to fluctuate with it. This also means that it will have to be repeatedly distributed throughout the area, wasting bandwidth and using extra CPU cycles on all of the routers. Conversely, if the lowest cost route is stable, the summary will be completely stable as well—even if all of the other summarized routes are unstable.

```
Router2#show ip route ospf
     172.16.0.0/24 is subnetted, 1 subnets
O IA    172.16.2.0 [110/140] via 10.1.1.2, 00:09:28, Serial0/0
O IA 172.20.0.0/16 [110/1692] via 10.1.1.2, 00:00:42, Serial0/0
O IA 172.25.0.0/16 [110/131] via 10.1.1.2, 00:00:25, Serial0/0
Router2#
```

In fact, the situation we just described, in which the ABR adopts the lowest summarized cost for the summary route, is not the RFC standard method. This method was the standard in RFC 1583. But, when OSPF was updated in RFC 2178 (and subsequently updated again in RFC 2328), the rule changed. In these newer versions of the OSPF standard, the rule is that the summary route should have the same cost as the highest-cost summarized route.

Naturally, there is the potential for serious problems in OSPF networks if some routers use the RFC 1583 rule and others use the newer rule. So, when the new standard came out, Cisco kept the old rule as the default, but included a command to allow you to easily change to the new standard:

```
Router1(config)#router ospf 55
Router1(config-router)#no compatible rfc1583
```

This command is available in IOS level 12.0 and higher. We recommend using caution when migrating a network from the old system of summarization to the new one. It is relatively common in OSPF networks to have several ABR routers to connect an area to Area 0. If you changed one of these ABRs to use the new summarization method, it would automatically have a worse metric than any of the other ABRs. All of the routers in the area would stop using the summary route pointing to this new style ABR in favor of the routes distributed by the remaining RFC 1583 ABRs. This has the potential to be extremely disruptive to a network, so you have to migrate all of the ABRs for an area at the same time.

The issue we discussed earlier about the stability of the individual route with the best metric now switches to concern about the stability of the route with the worst metric.

It's interesting to look at these summary routes on the ABR, which is responsible for doing all of the summarization. The ABR also includes the summary routes, but they aren't real routes, so it simply points them to its `Null0` interface. Cisco calls these *discard routes*. They help to prevent routing loops during summarization:

```
Router1#show ip route ospf
     172.20.0.0/16 is variably subnetted, 4 subnets, 4 masks
O       172.20.10.0/24 [110/1572] via 172.20.1.2, 00:07:42, Serial0/0.2
O      172.20.0.0/16 is a summary, 00:07:42, Null0
O       172.20.100.1/32 [110/1563] via 172.20.1.2, 00:07:42, Serial0/0.2
     172.25.0.0/16 is variably subnetted, 4 subnets, 3 masks
O       172.25.25.6/32 [110/11] via 172.25.1.7, 00:07:42, FastEthernet0/0.1
O      172.25.0.0/16 is a summary, 00:07:42, Null0
     10.0.0.0/8 is variably subnetted, 5 subnets, 4 masks
O       10.2.2.2/32 [110/1563] via 10.1.1.1, 00:07:42, Serial0/1
O      10.0.0.0/8 is a summary, 00:07:42, Null0
Router1#
```

Before IOS level 12.1(6), the only way to generate a discard route was to manually create a static route:

```
Router1(config)#ip route 172.20.0.0 255.255.0.0  null0
```

However, in IOS levels 12.1(6) and higher, this discard route is generated by default, and you don't need to create it. If you want to disable creation of the discard route, use the *no discard-route* command:

```
Router1(config)#router ospf 55
Router1(config-router)#no discard-route internal
Router1(config-router)#no discard-route external
```

With the *internal* keyword, this command prevents the router from automatically generating discard routes for internal summary routes. Similarly, the *external* keyword is for external routes. However, we urge caution with this command because the absence of a discard route can cause loops.

8.11.4 See Also

RFC 1583, RFC 2328

Top

Recipe 8.12 Disabling OSPF on Certain Interfaces

8.12.1 Problem

You want to prevent some of a router's interfaces from taking part in OSPF.

8.12.2 Solution

The *passive-interface* configuration command effectively disables OSPF on an interface by preventing it from forming OSPF adjacencies:

```
Router3#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router3(config)#router ospf 44
Router3(config-router)#network 0.0.0.0 255.255.255.255  area 100
Router3(config-router)#passive-interface Ethernet0
Router3(config-router)#end
Router3#
```

8.12.3 Discussion

OSPF will not start to exchange any routing information until two routers on a segment have authenticated (if authentication is enabled) and agreed on the various area parameters. So simply preventing one router from taking part in this handshake is sufficient to prevent the exchange of OSPF information on the interface. Also, while you can use a *passive-interface* command as shown in the example, you can also prevent an interface from taking part in OSPF by just using more restrictive *network* commands. In the example, the *network* statement includes everything. But you could just as easily use a network statement that restricts OSPF to a list of specific interfaces as follows:

```
Router3#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router3(config)#router ospf 44
Router3(config-router)#network 172.20.1.2 0.0.0.0 area 100
Router3(config-router)#network 172.20.10.1 0.0.0.0 area 100
Router3(config-router)#end
Router3#
```

Any interfaces that aren't explicitly included by a network statement will not take part in OSPF. On the other hand, sometimes a router can have a large number of interfaces, and you want all but one or two of them to take part in OSPF. In this case, it is more convenient to use passive interface commands.

To see the effect of this command, we'll look at a network both with and without the passive interface configured. Here is the neighbor list before configuring any passive interfaces:

```
Router3#show ip ospf neighbor

Neighbor ID     Pri   State           Dead Time   Address         Interface
172.20.220.1      1   FULL/BDR        00:00:39    172.20.10.2     Ethernet0
172.25.25.1       1   FULL/  -        00:00:37    172.20.1.1      Serial0.1
Router3#
```

Then, after making the `Ethernet0` interface passive, the router drops all of the neighbor relationships on this interface. We are left with only one neighbor:

```
Router3#show ip ospf neighbor

Neighbor ID     Pri   State           Dead Time   Address         Interface
172.25.25.1       1   FULL/  -        00:00:38    172.20.1.1      Serial0.1
Router3#
```

Of course, this also affects any routes that point to neighboring routers through this interface. This is the routing table before configuring `Ethernet0` as passive:

```
Router3#show ip route ospf
     172.20.0.0/16 is variably subnetted, 5 subnets, 3 masks
O       172.20.220.1/32 [110/11] via 172.20.10.2, 00:00:02, Ethernet0
O       172.20.200.1/32 [110/11] via 172.20.10.2, 00:00:02, Ethernet0
O*IA 0.0.0.0/0 [110/3572] via 172.20.1.1, 00:00:02, Serial0.1
Router3#
```

With the passive interface configured, all of the corresponding routes are also gone:

```
Router3#show ip route ospf
O*IA 0.0.0.0/0 [110/3572] via 172.20.1.1, 00:01:53, Serial0.1
Router3#
```

Top

Recipe 8.13 OSPF Route Tagging

8.13.1 Problem

You want to tag specific routes to prevent routing loops during mutual redistributing between routing protocols.

8.13.2 Solution

You can tag external routes in OSPF by using the *redistribute* command with the *tag* keyword:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#router ospf 55
Router1(config-router)#redistribute eigrp 11 metric-type 1 subnets tag  67
Router1(config-router)#end
Router1#
```

8.13.3 Discussion

Route tagging in OSPF is similar to route tagging in RIP Version 2 and EIGRP, which we discussed in Chapter 6 and Chapter 7. Just like those protocols, OSPF doesn't directly use the route tags. But they are useful when distributing routes into foreign routing protocols.

In the example configuration, Router1 is an ASBR that connects to a network that uses EIGRP process number 11. We have configured this router so that it redistributes these EIGRP routes into OSPF as external Type 1 routes with a tag value of 67:

```
Router5#show ip route 10.2.2.0
Routing entry for 10.2.2.0/30
  Known via "ospf 87", distance 110, metric 45
  Tag 67, type extern 1
  Redistributing via ospf 87
  Last update from 172.25.1.5 on Ethernet0, 00:07:14 ago
  Routing Descriptor Blocks:
  * 172.25.1.5, from 172.25.25.1, 00:07:14 ago, via Ethernet0
      Route metric is 45, traffic share count is 1
Router5#
```

The tags become useful when you redistribute the tagged routes into another network. For example, the following configuration shows how we might redistribute this group of external routes into RIP, but no

internal OSPF routes. This sort of configuration is useful if you want to allow the RIP and EIGRP external networks to talk to one another through your OSPF network, but prevent them from seeing your own routing tables:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#router rip
Router1(config-router)#version 2
Router1(config-router)#redistribute ospf 87 route-map  TAGGEDROUTES
Router1(config-router)#exit
Router1(config)#route-map TAGGEDROUTES permit 10
Router1(config-route-map)#match tag 67
Router1(config-route-map)#exit
Router1(config)#route-map TAGGEDROUTES deny 20
Router1(config-route-map)#end
Router1#
```

8.13.4 See Also

Chapter 6; Chapter 7

Top

Recipe 8.14 Logging OSPF Adjacency Changes

8.14.1 Problem

You want to monitor OSPF adjacency state changes to ensure network stability.

8.14.2 Solution

The *log-adjacency-changes* configuration command instructs the router to create a log message whenever two OSPF routers establish or break their adjacency relationship:

```
Router2#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router2(config)#router ospf 12
Router2(config-router)#log-adjacency-changes
Router2(config-router)#end
Router2#
```

8.14.3 Discussion

No routes are exchanged between routers if they lose their adjacency relationship. Every time this relationship is lost, the corresponding routes are removed and every router in the area must be updated with the new network topology. So it can be extremely useful to log these changes for troubleshooting, as well as for reconstructing serious problems that occurred in the past. We recommend using this option.

Here are some example log messages. The first message shows that the adjacency has been lost due to an expired timer. This means that this router has not heard its neighbor's regularly scheduled "hello" messages recently, so it needs to delete its routes. A few minutes later, the neighbor has come back and reestablished its adjacency:

```
Oct 14 09:54:13: %OSPF-5-ADJCHG: Process 12, Nbr 172.25.25.1 on Serial0/0 from FULL
to DOWN, Neighbor Down: Dead timer expired
Oct 14 09:57:43: %OSPF-5-ADJCHG: Process 12, Nbr 172.25.25.1 on Serial0/0 from
LOADING to FULL, Loading Done
```

Starting in 12.1, Cisco added the keyword*detail* :

```
Router2#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router2(config)#router ospf 12
Router2(config-router)#log-adjacency-changes detail
Router2(config-router)#end
```

```
Router2#
```

When you enable the detailed logging, you get considerably more information. It now shows all of the various stages that OSPF neighbors need to go through to establish their adjacencies:

```
%OSPF-5-ADJCHG: Process 12, Nbr 172.25.25.1 from FULL to DOWN, Neighbor Down: Dead
timer expired
%OSPF-5-ADJCHG: Process 12, Nbr 172.25.25.1 from DOWN to INIT, Received Hello
%OSPF-5-ADJCHG: Process 12, Nbr 172.25.25.1 from INIT to 2WAY, 2-Way Received
%OSPF-5-ADJCHG: Process 12, Nbr 172.25.25.1 from 2WAY to EXSTART, AdjOK?
%OSPF-5-ADJCHG: Process 12, Nbr 172.25.25.1 from EXSTART to EXCHANGE, Negotiation
Done
%OSPF-5-ADJCHG: Process 12, Nbr 172.25.25.1 from EXCHANGE to LOADING, Exchange Done
%OSPF-5-ADJCHG: Process 12, Nbr 172.25.25.1 from LOADING to FULL, Loading Done
```

This level of detail is rarely required unless you suspect that there is a problem with the handshake process between two routers. In that case it might be more effective to use debugging (as discussed in Recipe 8.17). We wouldn't normally recommend using the *detail* option in production networks because it just fills up the logs with extra messages.

8.14.4 See Also

Recipe 8.17

Top

Recipe 8.15 Adjusting OSPF Timers

8.15.1 Problem

You want to change the default OSPF timers to improve stability or convergence behavior.

8.15.2 Solution

You can improve the convergence time of OSPF on a particular interface by reducing the hello and dead timers:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#interface Serial0/1
Router1(config-if)#ip ospf hello-interval 5
Router1(config-if)#ip ospf dead-interval 20
Router1(config-if)#end
Router1#
```

If you make this change on one router, you must make it on all of the other routers sharing the same network segment:

```
Router2#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router2(config)#interface Serial0/0
Router2(config-if)#ip ospf hello-interval 5
Router2(config-if)#ip ospf dead-interval 20
Router2(config-if)#end
Router2#
```

8.15.3 Discussion

OSPF uses two timers. The hello timer controls how often the router sends routine messages to its neighbors simply indicating that it is still up. If the neighbors don't hear any hello messages for the length of time defined by the *dead-interval*, they will assume that the router is no longer reachable and drop it from the adjacency table.

The default values are 10 seconds for the hello time, and 40 seconds for the dead time. The usual rule of thumb with OSPF is to keep the dead time value four times the hello interval. However, this is not a strict rule. EIGRP, for example, uses a dead time that is only three times its hello interval. So, if you wanted

OSPF to have convergence times that more closely matched those of EIGRP, you could set the OSPF hello time to 5 seconds, and the dead interval to 15 seconds. Bear in mind that shortening the hello timer will increase the amount of traffic on the link. And shortening the dead interval increases the chances of losing adjacency because of network congestion or link errors.

It is important to adjust the timers on all of the routers on the network segment together. Unlike EIGRP, which allows every router to use a different set of timers, in OSPF the routers cannot establish adjacencies if their timers do not match exactly. But you can adjust the timers separately on different interfaces on a router. So you can use slower timers on low-bandwidth links, and faster timers on faster links. In general we don't recommend increasing the timers from their default values, but it can be useful to decrease them to improve convergence on important high-speed segments.

You can see the new timers with the *show ip ospf interface* command:

```
Router2#show ip ospf interface Serial0/0
Serial0/0 is up, line protocol is up
  Internet Address 10.1.1.1/30, Area 2
  Process ID 12, Router ID 192.168.30.1, Network Type POINT_TO_POINT, Cost: 130
  Transmit Delay is 1 sec, State POINT_TO_POINT,
  Timer intervals configured, Hello 5, Dead 20, Wait 20, Retransmit 5
    Hello due in 00:00:04
  Index 1/1, flood queue length 0
  Next 0x0(0)/0x0(0)
  Last flood scan length is 1, maximum is 1
  Last flood scan time is 0 msec, maximum is 0 msec
  Neighbor Count is 1, Adjacent neighbor count is 1
    Adjacent with neighbor 172.25.25.1
  Suppress hello for 0 neighbor(s)
  Simple password authentication enabled
Router2#
```

Looking at the neighbor table, you can see that the dead time reflects the change. Note that the time indicated in this output is the actual time remaining before OSPF declares this neighbor invalid. So, if everything is working properly, this value should count down from the configured dead time value until the hello interval expires. Then another hello packet will be sent and the dead timer will start over at its maximum value:

```
Router2#show ip ospf neighbor

Neighbor ID     Pri   State           Dead Time   Address         Interface
172.25.25.1       1   FULL/  -         00:00:19    10.1.1.2        Serial0/0
Router2#
```

8.15.4 See Also

Chapter 7

Top

Recipe 8.16 Viewing OSPF Status with Domain Names

8.16.1 Problem

You would prefer to view proper domain names in your OSPF show commands rather than the raw IP addresses.

8.16.2 Solution

You can configure OSPF to resolve IP addresses into router names with the following global configuration command:

```
Router3#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router3(config)#ip ospf name-lookup
Router3(config)#end
Router3#
```

8.16.3 Discussion

When you configure OSPF name lookup, the router will use its locally configured host table, if it has one, or DNS to resolve the names. If both are present, the router will check the local host table first. You can enable DNS on a router with the *ip domain-lookup* and *ip name-server* commands, as discussed in Chapter 2.

Enabling name resolution can be useful when displaying information such as OSPF neighbor tables. For example, if we look at the neighbor table without name lookup enabled, we see IP addresses:

```
Router3#show ip ospf neighbor

Neighbor ID     Pri   State           Dead Time   Address         Interface
172.20.220.1      1   FULL/DR         00:00:34    172.20.10.2     Ethernet0
172.25.25.1       1   FULL/  -        00:00:31    172.20.1.1      Serial0.1
Router3#
```

But, when name lookup is enabled, the router replaces the router IDs with names:

```
Router3#show ip ospf neighbor

Neighbor ID     Pri   State           Dead Time   Address         Interface
Router6           1   FULL/DR         00:00:37    172.20.10.2     Ethernet0
```

```
Router1           1   FULL/  -        00:00:36    172.20.1.1      Serial0.1
Router3#
```

8.16.4 See Also

Recipe 8.8; Chapter 2

Top

Recipe 8.17 Debugging OSPF

8.17.1 Problem

OSPF is not behaving properly and you want to debug it to isolate and solve the problem.

8.17.2 Solution

There are several OSPF debugging options. The most common symptoms of OSPF problems are instabilities in the neighbor relationships. So the most useful debugging option traces the formation of adjacencies:

```
Router3#debug ip ospf adj
OSPF adjacency events debugging is on
Router3#
```

8.17.3 Discussion

This particular debug output is especially useful because it helps to diagnose problems when routers refuse to form adjacencies with one another. For example, the following debug message indicates that there is an authentication problem. The neighbor router in this case is configured for MD5 authentication, while this router is configured for no authentication:

```
Dec 21 16:00:14.341: OSPF: Rcv pkt from 172.25.1.7, FastEthernet0/0.1 : Mismatch
Authentication type. Input packet specified type 2, we use type 0
```

Top

Chapter 9. BGP

Top

Introduction

Border Gateway Protocol (BGP) Version 4 is the lifeblood of the Internet. It is responsible for exchanging routing information between all of the major Internet Service Providers (ISPs), as well between larger client sites and their respective ISPs. And, in some large enterprise networks, BGP is used to interconnect different geographical or administrative regions.

Primarily to support the complexity of the public Internet, Cisco has added several clever and useful features to its BGP implementation. This book is focused on solutions to real-world problems, so we will not try to describe all of these features. And it would take a whole book to describe how to operate BGP in a large ISP network, so we will avoid discussing extremely large-scale BGP problems. Instead, we will look at two main classes of BGP problems: connecting a network to the public Internet, and interconnecting two or more Interior Gateway Protocols (IGP) in a private network.

A detailed discussion of the BGP protocol and its features is out of the scope of this book. For this type of information, we recommend referring instead to IP Routing by Ravi Malhotra (O'Reilly), or BGP by Iljitsch van Beijnum (O'Reilly). However, we will include a brief review of the most critical concepts.

Basic Terminology

BGP is an Exterior Gateway Protocol (EGP), which means that it exchanges routing information between Autonomous Systems (ASes). This is different from purely IGPs, such as RIP, EIGRP, and OSPF, which we discussed in Chapter 6, Chapter 7 and Chapter 8, respectively. It also uses a different basic algorithm for building a loop-free topology than any of those protocols. RIP is a distance vector protocol, OSPF is a link state protocol, and EIGRP is a distance vector protocol that incorporates many of the advantages of a link state protocol. BGP, on the other hand, uses a path vector algorithm. This means that instead of reducing each route's relative importance in the routing table to a single metric or cost value, BGP keeps a list of every AS that the path passes through. It uses this list to eliminate loops, because a router can check whether a route has already passed through a particular AS by simply looking at the path.

RFC 1930 describes what the Internet Engineering Task Force (IETF), which is the official Internet standards organization, considers to be the Best Current Practices (BCPs) for creating and numbering ASes. This document defines an AS as "a connected group of one or more IP prefixes run by one or more network operators which has a single and clearly defined routing policy." In practical terms, what appears on the Internet as a single AS may in fact represent an ISP as well as all their customer networks that aren't using BGP to advertise themselves as unique administrative domains.

A consistent routing policy in this context means that if a device on the edge of the AS advertises that it can handle routing for a particular set of prefixes, then all of the routers in the same AS can handle the same prefixes. It doesn't matter if some of these prefixes refer to internal routes and others refer to external routes. What matters is that the routers inside the AS must agree with one another on how to handle each route, and which internal or external router is the best place to send traffic for this particular network. This agreement is what it means for the AS to behave consistently.

It is important to note that this definition doesn't mean that there has to be one and only one IGP inside of an AS. In fact, there could be many IGPs, and there could even be no IGP. The interior routing inside of the AS could be handled entirely by a combination of BGP and static routes, for example.

BGP routers talk to one another over a permanent TCP connection on port 179. When BGP operates between two routers that are in the same AS, it is called Interior Border Gateway Protocol (iBGP). When the peers are in different ASes, they use external Border Gateway Protocol (eBGP). Unless you are using one of the more complex features that were invented specifically to avoid it, all of the BGP routers in an AS must peer with one another in a complete mesh. This ensures that the AS behaves consistently when advertising routes to other ASes.

Synchronization is a concept that comes up frequently in BGP configurations. Because the AS needs to behave consistently, if you run an IGP and iBGP, they have to agree. Think of a network where the iBGP peers are several hops apart, and the intervening network uses an IGP to communicate between them. Synchronization requires that, for a BGP route to be useable, the IGP must also contain a route to the same prefix. This ensures that one of these BGP peer routers doesn't try to forward a packet to the other internal BGP peer unless the network connecting them knows what to do with this packet.

Cisco routers allow you to disable synchronization, which is actually necessary in any case where you don't redistribute the IGP routes into BGP. Make sure that your network design doesn't require the IGP to have access to the BGP routes in order to communicate between the iBGP peers if you disable synchronization.

Every discussion of BGP includes frequent references to IP *prefixes*. A prefix is a Classless Inter-Domain Routing (CIDR) block of addresses. We previously discussed CIDR in Chapter 5. CIDR is a set of rules for IP subnetting that allows you to summarize groups of IP addresses. For example, you might have four network segments that use the IP addresses `172.25.4.0/24`, `172.25.5.0/24`, `172.25.6.0/24` and `172.25.7.0/24`. Each of these network addresses is a prefix. If, for example, you wanted to send a packet to the device `172.25.5.5/32`, your router only needs to know how to route packets for `172.25.5.0/24`. This route prefix includes the specific host address.

But you can go one step further than this. If the paths to all of these IP networks pass through the same router, it is often useful to summarize or aggregate the prefixes. The router that leads to all of these networks might simply advertise a single prefix, `172.25.4.0/22`, that covers all of the individual networks.

Similarly, CIDR allows you to create supernets that summarize several classful networks. For example, you could summarize `172.24.0.0/16` through `172.31.0.0/16` as `172.24.0.0/13`.

BGP requires that every AS must have a 16-bit Autonomous System Number (ASN). Because it is 16 bits long, the ASN can have any value between and 65535.

The ASN is a globally unique identification number. BGP uses these ASNs to eliminate loops. Suppose two networks are using the same ASN. A router in the first AS will send out its routes normally, but the BGP router for the second network will drop these routes because they already appear to have passed through this AS. So it is important to ensure that you follow the rules on ASN selection, which are described in RFC 1930.

RFC 1930 originally divided up the range from 1 through 22,527 among the three major international Internet registry organizations (RIPE, ARIN, and APNIC) to allocate to networks connected to the public Internet. Since publication of that RFC, however, the IANA has distributed further blocks of numbers, currently extending up to almost ASN 30,000.

Just as RFC 1918 defines private unregistered ranges of IP addresses for networks that don't connect directly to the public Internet, RFC 1930 defines a series of private unregistered ASN values. You can use these private ASNs freely as long as they don't leak onto the public Internet. And, just as you can use NAT to hide your private IP addresses when you connect to the Internet, you can also hide private ASNs, as long as the AS that connects directly to the public Internet has a registered ASN and registered IP addresses.

All ASN values between 64,512 and 65,534 are designated for private use. This gives 1,023 ASN values that you can freely use in your internal network without registering, and without fear of conflict. If you use these private ASNs in an enterprise network, you must ensure that each private ASN is unique throughout the network. Enterprise networks that are large enough to require multiple ASs are generally managed by several different groups, so it is critical to coordinate the use of these private ASNs. If there is ever a conflict, with two ASes using the same ASN, it will disrupt routing to both of the conflicting ASes. And, if either of the conflicting ASes is used for transit, it could disrupt routing throughout the entire enterprise network, causing routing loops and unreachable networks. Although, of course, each individual AS will continue to function normally internally.

There are many situations where you can use unregistered ASNs. In fact, the only time a registered ASN is required is when you need to use BGP to exchange routing information with an ISP. Note that if you only have a single link to a single ISP, then you really don't require BGP at all. If you have only a single connection to the Internet, then you can get by with a single default route to the Internet because everything passes through this one link. If the link does go, there's nothing you can do anyway. So, in this case, running BGP is overkill. A small router with a default route is more than adequate.

You should consult your ISP to discuss your options. They might also be willing to let you use BGP and a private ASN, which they will remove when passing your routes to the rest of the world. They may

even be willing to let you run a simpler routing protocol (such as RIPv2) to provide redundancy among two or more links that all use their network. In any case, your ISP will probably not pass your routes directly to the Internet anyway. It is more likely, and preferable, that they will allocate addresses to your network from a range that they can summarize. Then the ISP will just pass a single routing entry to the rest of the Internet to represent many customer networks.

You can also do this kind of AS path filtering internally. If you have several internal ASs, only one of which connects to the public Internet, then you can register the one directly connected ASN, and simply filter the private ASNs out of any path information that you pass to your ISPs. We show an example of this kind of filtering in Recipe 9.9.

Another special ASN value that bears mentioning is 65,535, which the IANA reserves for future requirements. RFC 1930, on the other hand, says that this ASN is part of the range that is freely available for unregistered use. We recommend avoiding this number, because the IANA is the ultimate authority. Although there is currently no conflict with this number, the IANA may decide to give it some special significance later, which could break existing private networks that might use it.

Throughout this chapter we will use private ASNs and private IP addresses in examples that are intended to represent the public Internet. This is purely for demonstration purposes. You must never allow these private values to reach the public Internet.

BGP Attributes

BGP associates several different basic attributes with each route prefix. These attributes include useful pieces of information about the route, where it came from, and how to reach it. *Well-known attributes* must be supported by every BGP implementation. Some well-known attributes are mandatory. All of the *mandatory attributes* must be included with every route entry. A BGP router will generate an error message if it receives a route that is missing one or more well-known mandatory attributes.

There are also well-known *discretionary attributes*, which every BGP router must recognize and support, but they don't have to be present with every route entry. Whenever a router passes along a route that it has learned via BGP to another BGP peer, it must include all of the well-known attributes that came with this route, including any discretionary attributes. Of course, the router may need to update some of these attributes before passing them along, to include itself in the path, for example.

BGP routes can also include one or more *optional attributes*. These are not necessarily supported by all BGP implementations. Optional attributes can be either *transitive* or *non-transitive*, which is specified by a special flag in the attribute type field. If a router receives a route with a transitive optional attribute, it will pass this information along intact to other BGP routers, even if it doesn't understand the option. The router will mark the Partial bit in the attribute flags to indicate that it was unable to handle this attribute, however.

The router will quietly drop any unrecognized non-transitive optional attributes from the route information without taking any action.

We will now describe several of the most common BGP attributes.

ORIGIN (well-known, mandatory)

This attribute can have one of three different values, reflecting how the BGP router that was responsible for originating the route first learned of it. The possible values are:

0 - IGP: The route came from an IGP interior to the originating AS.

1 - EGP: The route came from an EGP other than BGP.

2 - Incomplete: Any other method.

AS_PATH (well-known, mandatory)

The AS_PATH is a list of ASNs, showing the path taken to reach the destination network. There are actually two types of AS_PATHs. An AS_SEQUENCE describes the literal path taken to reach the destination, while an AS_SET is an unordered list of ASNs along the path. Each time a BGP router passes a route update to an eBGP peer, it updates the AS_PATH variable to include its own ASN.

NEXT_HOP (well-known, mandatory)

This attribute carries the IP address of the first BGP router along the path to the destination network. When the router installs the route for the associated prefix in its routing table, it will use this attribute for the next-hop router. This is where the router will forward its packets for this destination network.

By default, the NEXT_HOP router will be the router that announced this route to the AS. For routes learned from an external AS via eBGP, the NEXT_HOP router will be the first router in the neighboring AS. This information is passed intact throughout the AS using iBGP, so all routers in the AS see the same NEXT_HOP router.

MULTI_EXIT_DISC (optional, non-transitive)

The Multiple Exit Discriminatory (MED) option is also often called the BGP Metric. Because this 32-bit value is non-transitive, it is only propagated to adjacent ASes. Routers can use the MED to help differentiate between two or more equivalent paths between these ASes.

LOCAL_PREF (well-known, mandatory)

BGP only distributes Local Preference information with routes inside of an AS. Routers can use this number to allow the network to favor a particular exit point to reach a destination network. This information is not included with eBGP route updates.

ATOMIC_AGGREGATE (well-known, discretionary)

When a BGP router aggregates several route prefixes to simplify the routing tables that it passes to its peers, it usually sets the ATOMIC_AGGREGATE attribute to indicate that some information has been lost. It doesn't set this attribute, however, in cases where it uses an AS_SET in its AS_PATH to show the ASNs of all of the different prefixes being summarized.

AGGREGATOR (optional, transitive)

The AGGREGATOR attribute indicates that a router has summarized a range of prefixes. The router doing the route aggregation can include this attribute, which will include its own ASN and IP address or router ID.

Both the AGGREGATOR and the ATOMIC_AGGREGATE attributes have become relatively uncommon since the universal conversion to BGP Version 4.

COMMUNITY (optional, transitive)

A COMMUNITY is a logical grouping of networks. This attribute is defined in RFC 1997, and RFC 1998 describes a useful application of the concept to ISP networks.

MP_REACH_NLRI (optional, non-transitive)

This attribute carries information about reachable multiprotocol destinations and next-hop routers. Multiprotocol in this context could refer to any foreign protocol such as IPv6, although it is most commonly used with IP multicast, as we discuss in Chapter 23. MultiProtocol Label Switching (MPLS) also uses MBGP for per-VPN routing tables.

Carrying foreign routing information this way ensures backward compatibility. Routers that don't support the extension can easily interoperate with routers that do.

MP_UNREACH_NLRI (optional, non-transitive)

The MP_UNREACH_NLRI attribute is similar to the MP_REACH_NLRI, except that it carries information about unreachable multiprotocol destinations.

BGP has several other optional attributes as well, although we will not discuss them in this book. For more information, we suggest referring to Internet Routing Architectures (Cisco Press).

Route Selection

Unlike the various interior routing protocols that we discussed in the preceding chapters, BGP doesn't support *multipath* routing by default. So, if there are two or more paths to a destination, BGP will go to great extremes to ensure that only one of them is actually used.

BGP decides which route to use by applying a series of tests in order. It is important to understand these tests and the order that the router looks at them, particularly when you are trying to influence which routes are used. Otherwise you might end up wasting a lot of time trying to adjust your routing tables using one method, while the router is making the actual decision at some earlier step, without

ever seeing your adjustments.

Note that at each step, there may be several routes to the same destination prefix that all meet the requirement, or are equal after a particular test. In that case, BGP will proceed to the next test to attempt to break the tie.

We should point out that these are the route selection rules on Cisco routers. Several of these rules are not part of the BGP specification. So, for non-Cisco equipment, you should consult the vendor's BGP documentation to see what the differences are.

1. The first test is whether the next-hop router is accessible. By default, routers do not update the next-hop attribute when exchanging routes by iBGP. So it is possible to receive a route whose next-hop router is actually several hops away, and perhaps unreachable. BGP will not pass these routes to the main routing table, but it will keep them in its own route database.

2. If synchronization is enabled, the router will ignore any iBGP routes that are not synchronized.

3. The third test uses the Cisco proprietary *weight* parameter, selecting the route with the largest weight. This parameter is not part of the routing protocol. Adjusting the weight of a particular route on a router will only affect route selection on this router. It is a purely local concept. The default weight value is zero, except for locally sourced routes, which get a default weight of 32,768. The maximum possible weight is 65,535.

4. If the weights are the same, BGP then selects the route with the highest Local Preference value from the LOCAL_PREF attribute. Routers only include this attribute when communicating within an AS (iBGP). For external routes, the router that receives a particular route via eBGP sets the Local Preference value. For internal routes, it is set by the router that introduced the route into BGP. This allows you to force every router in your AS to preferentially send traffic for a particular destination through a particular eBGP link.

5. If two or more routes to the same destination network are still equal after comparing Local Preference values, the router moves on to look at the AS_PATH. This is the path vector that gives BGP its essential character. It is a set of AS numbers that describes the path to the destination network.

   BGP routers prefer routes that originate inside their own ASes.

   For routes that originate outside of the AS, BGP will prefer the one with the shortest path (i.e., the one with the fewest ASNs). This is a simple indication of the most direct path.

6. BGP then looks at the ORIGIN attribute if the AS path lengths are the same, and selects IGP routes in preference to EGP, and EGP in preference to INCOMPLETE routes. An INCOMPLETE route is one that is injected into BGP via redistribution, so BGP isn't able to vouch for its validity.

7. The next test looks at the MED, and selects the route with the lowest value. The MED is used only if both routes are received from the same AS, or if the command *bgp always-compare-med* has been enabled. With this command enabled, BGP will compare MED values even if they come from different ASes, although to reach this step the AS_PATHs must have the same length. Note that if you use this command at all, you should use it throughout the AS or you risk creating routing loops. MED values are only propagated to adjacent ASes, so routers that are further downstream don't see them at all.

8. BGP prefers eBGP to iBGP paths. This helps to eliminate loops by ensuring that the route selected is the one that leads out of the AS most directly. Note that the iBGP routes don't include internal routes that are sourced from within your AS, because they are selected at step number 5. So this test only looks at routes to external destinations.

9. The next test compares the IGP costs of the paths to the next-hop routers, and selects the closest one. This helps to ensure that faster links and shorter paths are used where possible.

10. Next, BGP will look at the ages of the routes and use the oldest route to a particular destination. This is an indication of stability. If two routes are otherwise equivalent, it is best to use the one that appears to be the most stable.

11. Finally, if the routes are still equivalent, BGP resorts to the router IDs of the next-hop routers to break any ties, selecting the next-hop router with the lowest router ID. Since router IDs are unique, this is guaranteed to eliminate any remaining duplicate route problems.

Note that there are subtle variations to these rules for special situations such as AS confederations, and many individual rules can be disabled if you want the router to skip them.

Cisco has also implemented a BGP multipath option that changes this route selection process somewhat. If you enable multiple path support, BGP will still perform the first 7 tests, evaluating everything up to and including the MED values. But, if two or more routes are still equivalent at this point, the router will install some or all of them (depending on how you implement this feature). Please refer to Recipe 9.8 for a discussion of this option.

Top

Recipe 9.1 Configuring BGP

9.1.1 Problem

You want to run BGP in a simple network.

9.1.2 Solution

In its simplest configuration, BGP exchanges routes between a router in one AS and another router in a different AS. The first router is in AS 65500:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#interface Serial0
Router1(config-if)#ip address 192.168.55.6 255.255.255.252
Router1(config-if)#exit
Router1(config)#router bgp 65500
Router1(config-router)#network 192.168.1.0
Router1(config-router)#neighbor 192.168.55.5 remote-as 65501
Router1(config-router)#no synchronization
Router1(config-router)#end
Router1#
```

The second router is in AS 65501:

```
Router2#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router2(config)#interface Serial0
Router2(config-if)#ip address 192.168.55.5 255.255.255.252
Router2(config-if)#exit
Router2(config)#router bgp 65501
Router2(config-router)#network 172.25.17.0 mask 255.255.255.0
Router2(config-router)#neighbor 192.168.55.6 remote-as 65500
Router2(config-router)#no synchronization
Router2(config-router)#end
Router2#
```

9.1.3 Discussion

This example shows two routers in different ASes. Router1 is in AS 65500, and is configured to share routing information only for a single network using the command *network 192.168.1.0*. Because this is a classful network, we don't need to include a mask. However, notice that the syntax of the *network*

command on Router2 is different:

```
Router2(config-router)#network 172.25.17.0 mask 255.255.255.0
```

This is because the routing information we want to share only includes `172.25.17.0/24`, and not the entire classful network, `172.25.0.0/16`.

The first thing you should do after configuring two routers for BGP is to ensure that they are able to establish a BGP connection. You can verify this with the command *show ip bgp summary*:

```
Router1#show ip bgp summary
BGP router identifier 192.168.99.5, local AS number 65500
BGP table version is 7, main routing table version 7
4 network entries and 4 paths using 484 bytes of memory
2 BGP path attribute entries using 196 bytes of memory
BGP activity 11/7 prefixes, 11/7 paths

Neighbor        V    AS MsgRcvd MsgSent   TblVer  InQ OutQ Up/Down  State/PfxRcd
192.168.55.5    4 65501      17      18        7    0    0 00:12:38        2
Router1#
```

Here you can see that Router1 has a BGP neighbor,`192.168.55.5`, in AS 65501. The most critical detail here is the last column, "State/PfxRcd." In this column you will see either a word, indicating the state of the peer connection, or a number, indicating the number of routing prefixes (that is, the number of distinct subnets in the routing table) that have been received from this peer.

In this case, this router has had a valid BGP session with the neighbor device, `192.168.55.5` for just over 12 minutes. If this session is broken for any reason, you will most likely see either the word "Active" or "Idle" in this field. The following output shows another peer device, `172.25.2.2`, which is down:

```
Router1#show ip bgp summary
BGP router identifier 192.168.99.5, local AS number 65500
BGP table version is 7, main routing table version 7
4 network entries and 4 paths using 484 bytes of memory
2 BGP path attribute entries using 196 bytes of memory
BGP activity 11/7 prefixes, 11/7 paths

Neighbor        V    AS MsgRcvd MsgSent   TblVer  InQ OutQ Up/Down  State/PfxRcd
192.168.55.5    4 65501      17      18        7    0    0 00:12:38        2
172.25.2.2      4 65531     527     526        0    0    0 21:05:23 Active
Router1#
```

More than one engineer has seen the word "Active" (or "Connect") here and thought that the session was active. But, in fact, it means that this peer relationship is currently down. The BGP connection is only up if you see a number in the last column. Note also that the word "Idle" in this column indicates that the router doesn't believe that a session is even possible with this peer device, or that it has not yet attempted to connect (the router will wait several seconds before attempting a connection). If the Idle

condition persists, this usually indicates that the remote peer is unreachable. A persistent Active state, on the other hand, most likely indicates a configuration problem.

It often takes almost a minute to establish a BGP peer connection, so be patient if you don't see the peers immediately connect. If they have still failed to connect after this time, you should double-check your "neighbor" configuration statements. Make sure that you have the right remote IP address and AS number in particular. If these are correct, and you can ping the remote peer's IP address, then you should make sure that the routers are using the interfaces that you think they are to reach the destination.

The example in the solutions section of this recipe shows an eBGP peer relationship because we have configured different ASNs on the two routers:

```
Router1(config)#router bgp 65500
Router1(config-router)#neighbor 192.168.55.5 remote-as 65501
```

This shows that Router1 is in AS 65500, while Router2 is in AS 65501. You configure iBGP peers the same way, but the *neighbor* statement specifies the same ASN value as the *router bgp* statement. We can add a iBGP peer in AS 65500 as follows:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#interface Ethernet0
Router1(config-if)#ip address 192.168.1.5 255.255.255.0
Router1(config-if)#exit
Router1(config)#router bgp 65500
Router1(config-router)#neighbor 192.168.1.6 remote-as 65500
Router1(config-router)#end
Router1#
```

We would configure the other iBGP peer router like this:

```
Router3#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router3(config)#interface Ethernet0
Router3(config-if)#ip address 192.168.1.6 255.255.255.0
Router3(config-if)#exit
Router3(config)#router bgp 65500
Router3(config-router)#neighbor 192.168.1.5 remote-as 65500
Router3(config-router)#end
Router3#
```

There is no need to establish a peer relationship between this new router and the eBGP peer, Router2. Router3 may connect to one or more other, completely different ASes, though. And there is nothing to prevent you from having an iBGP peer that doesn't connect to any eBGP peers. However, it is important to create a full mesh of iBGP relationships among all of the BGP routers inside any given AS.

BGP uses a permanent TCP connection between pairs of peer routers, and every peer relationship must be configured manually. This is actually one of the biggest strengths of BGP, because it allows you to configure unique properties, such as unique filtering for each peer. With the various IGPs that we have already discussed, the routing peers discover one another dynamically by default.

The previous examples specify only the destination IP address, not the source address. In this particular case, there is only one way to reach the destination, so there is no need to specify the source address. The routers will simply use the IP address of the nearest interface. There are some cases where you do need to specify the source address, though.

For example, you might have two iBGP routers in your network, with several different possible paths between them. In this case, it would be better to configure the two routers to use their loopback addresses for the peer statements, rather than the physical interfaces, which could go down. If you have redundant paths, you may as well use them. You could configure the router to use its loopback address for BGP as follows:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#interface Ethernet0
Router1(config-if)#ip address 192.168.55.6 255.255.255.0
Router1(config-if)#exit
Router1(config)#interface Ethernet1
Router1(config-if)#ip address 192.168.56.10 255.255.255.0
Router1(config-if)#exit
Router1(config)#interface Loopback0
Router1(config-if)#ip address 172.21.19.1 255.255.255.255
Router1(config-if)#exit
Router1(config)#ip route 172.20.1.2 255.255.255.255 192.168.55.1
Router1(config)#ip route 172.20.1.2 255.255.255.255 192.168.56.1
Router1(config)#router bgp 65500
Router1(config-router)#neighbor 172.20.1.2 remote-as 65500
Router1(config-router)#neighbor 172.20.1.2 update-source Loopback0
Router1(config-router)#end
Router1#
```

Then, on the other router, you would have:

```
Router3#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router3(config)#interface Ethernet0
Router3(config-if)#ip address 192.168.55.1 255.255.255.0
Router3(config-if)#exit
Router3(config)#interface Ethernet1
Router3(config-if)#ip address 192.168.56.1 255.255.255.0
Router3(config-if)#exit
Router3(config)#interface Loopback0
Router3(config-if)#ip address 172.20.1.2 255.255.255.255
Router3(config-if)#exit
Router3(config)#ip route 172.21.19.1 255.255.255.255 192.168.55.6
```

```
Router3(config)#ip route 172.21.19.1 255.255.255.255 192.168.56.10
Router3(config)#router bgp 65500
Router3(config-router)#neighbor 172.21.19.1 remote-as 65500
Router3(config-router)#neighbor 172.21.19.1 update-source Loopback0
Router3(config-router)#end
Router3#
```

Each of these routers uses the other's loopback IP address for its BGP neighbor statement. But, to create a TCP session, you need the source address from one end to match the destination address of the other. So we have included commands to force each router to use their loopback interfaces for these source addresses:

```
Router1(config-router)#neighbor 172.20.1.2 update-source Loopback0
```

We strongly recommend using the *update-source* option and specifying a loopback interface on both routers whenever you have redundant paths between iBGP peers.

So far, everything that we have discussed has to do with establishing the iBGP and eBGP peer relationships. We haven't exchanged any actual routing information yet. This brings us to the *network* commands in the example configuration files. On the first router, we used the classful version of the command to advertise an entire Class C network, `192.168.1.0/24`.

```
Router1(config)#router bgp 65500
Router1(config-router)#network 192.168.1.0
```

The second router, however, uses the more general classless version of the *network* command:

```
Router2(config)#router bgp 65501
Router2(config-router)#network 172.25.17.0 mask 255.255.255.0
```

These commands allow the router to pick up routes out of its routing table and pass them along using BGP. BGP will not advertise anything that it doesn't have in its routing table. The first command will advertise the prefix `192.168.1.0/24` if it is in the routing table, while the second will advertise `172.25.17.0/24`. It is important to realize that these are literally the prefixes that BGP will advertise. If you have a route for `192.168.1.4/32`, then the first network statement we mentioned will not cover it. Instead, you would have to explicitly include a *network* command for this prefix:

```
Router1(config)#router bgp 65500
Router1(config-router)#network 192.168.1.4 mask 255.255.255.255
```

You can also use redistribution to inject routes into BGP from either static routes or foreign routing protocols. As we discuss in Recipe 9.14, however, redistribution is messy and complicated. We strongly recommend against redistribution to introduce routes into BGP in nearly all situations.

Because BGP will only advertise a prefix if it is in the routing table, an unstable IGP route could introduce instability into BGP. You can ensure that the route is always available, though, by using a floating static route pointing to the null interface:

```
Router1(config)#ip route 192.168.1.0 255.255.255.0  null0 250
```

Here we have specified an administrative distance of 250 for this route. This value is deliberately very high to ensure that it is worse than any IGP, as well as iBGP. Now, when the dynamic route drops out of the IGP routing table, the router will replace it with this floating static route, and BGP will continue to advertise the prefix. This is not always desirable, of course. You may want this BGP router to stop advertising routes that it cannot reach. But, in most cases, stability is more important. Please see Recipe 5.5 for more information about floating static routes.

Looking back at the example in the solutions section of this recipe, you will see that we disabled synchronization on both routers:

```
Router1(config)#router bgp 65500
Router1(config-router)#no synchronization
```

Synchronization is enabled by default. This feature is intended for situations where your AS acts as a transit for packets from one AS to another, but some of the routers in your AS do not run BGP. In such cases, the routers that only run the IGP need to have the same routing table as the BGP routers, or the AS could become a black hole for the unsynchronized routes. If synchronization is enabled in this situation, BGP will only advertise routes that are present in both the IGP and BGP route tables.

In this example, we had no intention of carrying the BGP routing table through the IGP. We recommend disabling synchronization unless you are running an IGP and redistributing routes between BGP and the IGP.

Take a close look at the examples in this recipe because they show how Cisco's BGP configuration syntax works. When you want to change the parameters for a particular peer, you must first define the neighbor and the AS that this peer resides in. Then you can start to define any non-default behavior for this peer with further *neighbor* commands that specify the same peer IP address. There are dozens of different options that you can adjust this way. We will mention several of these options in this chapter.

9.1.4 See Also

Recipe 5.5; Recipe 9.2; Recipe 9.14

Top

Recipe 9.2 Using eBGP Multihop

9.2.1 Problem

You want to use BGP to exchange routes with an external peer router that is more than one hop away. This situation can arise when the router at the edge of the network doesn't support BGP.

9.2.2 Solution

Cisco provides a useful option called eBGP multihop, which allows you to establish eBGP peer relationships between routers that are not directly connected to one another:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#ip route 172.20.1.2 255.255.255.255 192.168.1.5 2
Router1(config)#router bgp 65500
Router1(config-router)#neighbor 172.20.1.2 remote-as 65530
Router1(config-router)#neighbor 172.20.1.2 update-source Loopback0
Router1(config-router)#neighbor 172.20.1.2 ebgp-multihop
Router1(config-router)#end
Router1#
```

9.2.3 Discussion

In this example, we have only shown the configuration for one of the routers, although you will need to configure the *ebgp-multihop* keyword for the corresponding peer device as well.

This feature is not a standard part of the BGP protocol, but several router vendors implement it. The standard behavior requires eBGP routers to be adjacent to one another.

You might want to use this feature, for example, if the router at the edge of either your AS or the AS you are connecting to doesn't support BGP. The router will also need to have a route to the destination device, because it is not directly connected. We have included a static route for this purpose.

The *ebgp-multihop* keyword takes an optional argument, which can be any integer between 1 and 255. This represents the maximum number of hops between this router and the neighbor, which is used in the TTL field of the IP packet when establishing the peer connection. If you don't specify*ebgp-multihop*, the router will assume that the peers are adjacent and use a TTL value of 1. However, if you specify this keyword without an argument, the router will default to a TTL value of 255.

Note that you can cause some seriously strange routing problems using a high TTL value with this option. Suppose you have two ISPs, and your connection to one of them becomes unavailable. The routers could discover another path to one another, and reestablish their BGP peer relationship through the second ISP. This would cause extremely inefficient routing. You can avoid this problem by using static host routes and directing traffic for each peer router through the correct circuit.

In general, we recommend using the lowest possible value that still reaches the destination. However, the IETF has recently published the draft document, http://www.ietf.org/internet-drafts/draft-gill-btsh-02.txt, which describes another extremely interesting way of using this feature to improve security. The idea is that the only way that a packet can reach its destination with a TTL value of 254 or 255 is if the source is adjacent to the destination. If a more distant device were to attempt a BGP spoofing attack, the packets would arrive with a lower TTL value unless the attacker was also on a physically adjacent network.

This document suggests deliberately configuring your routers to use the highest possible TTL value. Then the routers would check the TTL value and discard any BGP packets with a TTL of less than 254. Unfortunately, we are not aware of any way to implement this idea on a Cisco router today because IP access lists do not include a way to filter based on TTL values. However, if this idea catches on, it seems likely that Cisco would provide a way to do this.

In the meantime, the best way to secure your BGP peers is to use MD5 authentication, as discussed in Recipe 9.16.

9.2.4 See Also

Recipe 9.16; The BGP TTL Security Hack (BTSH), http://www.ietf.org/internet-drafts/draft-gill-btsh-02.txt, December 2002

Top

Recipe 9.3 Adjusting the Next-Hop Attribute

9.3.1 Problem

You want to change the next-hop attribute on routes while distributing them via iBGP so that the routes always point to a next-hop address that is inside your AS.

9.3.2 Solution

By default, the value of the next-hop attribute for an external route is the IP address of the external BGP router that announced this route to the AS. You can change this behavior so that the next-hop router is an internal router instead by using the *next-hop-self* command:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#router bgp 65500
Router1(config-router)#neighbor 192.168.1.6 remote-as 65500
Router1(config-router)#neighbor 192.168.1.6 next-hop-self
Router1(config-router)#end
Router1#
```

9.3.3 Discussion

The next-hop attribute for a route depends on which router announces it. When a router passes route information to a peer in a different AS (using eBGP), it will generally update the next-hop attribute with its own IP address. However, by default iBGP peers will not change this attribute. For internal routes, the next-hop attribute will be the IP address of the router that sourced the internal route into BGP.

The result is that all of the routers inside of an AS will see the same external device as the next-hop router, even if that router is actually several physical hops away. The following output shows the BGP table of one of the routers in our AS before we specified the *next-hop-self* command. All of the next-hop addresses correspond to routers in other ASes:

```
Router2#show ip bgp
BGP table version is 10, local router ID is 11.5.5.1
Status codes: s suppressed, d damped, h history, * valid, > best, i - internal
Origin codes: i - IGP, e - EGP, ? - incomplete

   Network          Next Hop            Metric LocPrf Weight Path
```

```
<routes delete for brevity>
*>i172.22.1.0/24    172.25.1.7               0    100      0 65510 ?
* i172.24.0.0       172.22.1.3               0    100      0 ?
* i172.25.0.0       172.22.1.3               0    100      0 i
*>i172.25.2.0/30    172.25.1.7               0    100      0 65510 ?
*>i172.20.0.0/14    172.20.1.2                    100      0 65530 65501 ?
```

There will be serious routing problems if the router doesn't know how to reach one of these next-hop routers. And this is actually a distinct possibility because, for all external routes, these next-hop IP addresses will be in a different AS. Unless you use static routes or take pains to ensure that the IGP distributes all of these addresses, the other iBGP routers will not have a route to the next hop.

As we mentioned in the introduction to this chapter, the first thing that BGP checks when looking at routes is whether the next-hop router is reachable. Even if BGP didn't do this check, a route that has an unreachable next-hop router is clearly not going to be very useful.

However, you can use the *next-hop-self* command to configure a router to insert its own IP address in the next-hop attribute when passing routes to another router via iBGP:

```
Router1(config)#router bgp 65500
Router1(config-router)#neighbor 192.168.1.6 remote-as 65500
Router1(config-router)#neighbor 192.168.1.6 next-hop-self
```

Then the next hop of every route in the route table is guaranteed to be accessible:

```
Router2#show ip bgp
BGP table version is 10, local router ID is 11.5.5.1
Status codes: s suppressed, d damped, h history, * valid, > best, i - internal
Origin codes: i - IGP, e - EGP, ? - incomplete

   Network          Next Hop            Metric LocPrf Weight Path
<routes delete for brevity>
*>i172.22.1.0/24    192.168.1.6              0    100      0 65510 ?
* i172.24.0.0       192.168.1.6              0    100      0 ?
* i172.25.0.0       192.168.1.6              0    100      0 i
*>i172.25.2.0/30    192.168.1.6              0    100      0 65510 ?
*>i172.20.0.0/14    192.168.1.6                   100      0 65530 65501 ?
```

You can also configure this keyword for an eBGP peer, but it has no effect.

Note that you can also construct a route map to manually set the next-hop attribute to any IP address that you like. However, we don't recommend doing this as it too easy to make a mistake and forward routes with unreachable next-hop routers.

Top

Recipe 9.4 Connecting to Two ISPs

9.4.1 Problem

You want to set up BGP to support two redundant Internet connections.

9.4.2 Solution

The following configuration shows how to make the basic BGP connections, but it has serious problems that we will show how to fix in other recipes in this chapter:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#interface Serial0
Router1(config-if)#description connection to ISP #1, ASN 65510
Router1(config-if)#ip address 192.168.1.6 255.255.255.252
Router1(config-if)#exit
Router1(config)#interface Serial1
Router1(config-if)#description connection to ISP #2, ASN 65520
Router1(config-if)#ip address 192.168.2.6 255.255.255.252
Router1(config-if)#exit
Router1(config)#interface Ethernet0
Router1(config-if)#description connection to internal network, ASN 65501
Router1(config-if)#ip address 172.18.5.2 255.255.255.0
Router1(config-if)#exit
Router1(config)#router bgp 65501
Router1(config-router)#network 172.18.5.0 mask 255.255.255.0
Router1(config-router)#neighbor 192.168.1.5 remote-as 65510
Router1(config-router)#neighbor 192.168.2.5 remote-as 65520
Router1(config-router)#no synchronization
Router1(config-router)#end
Router1#
```

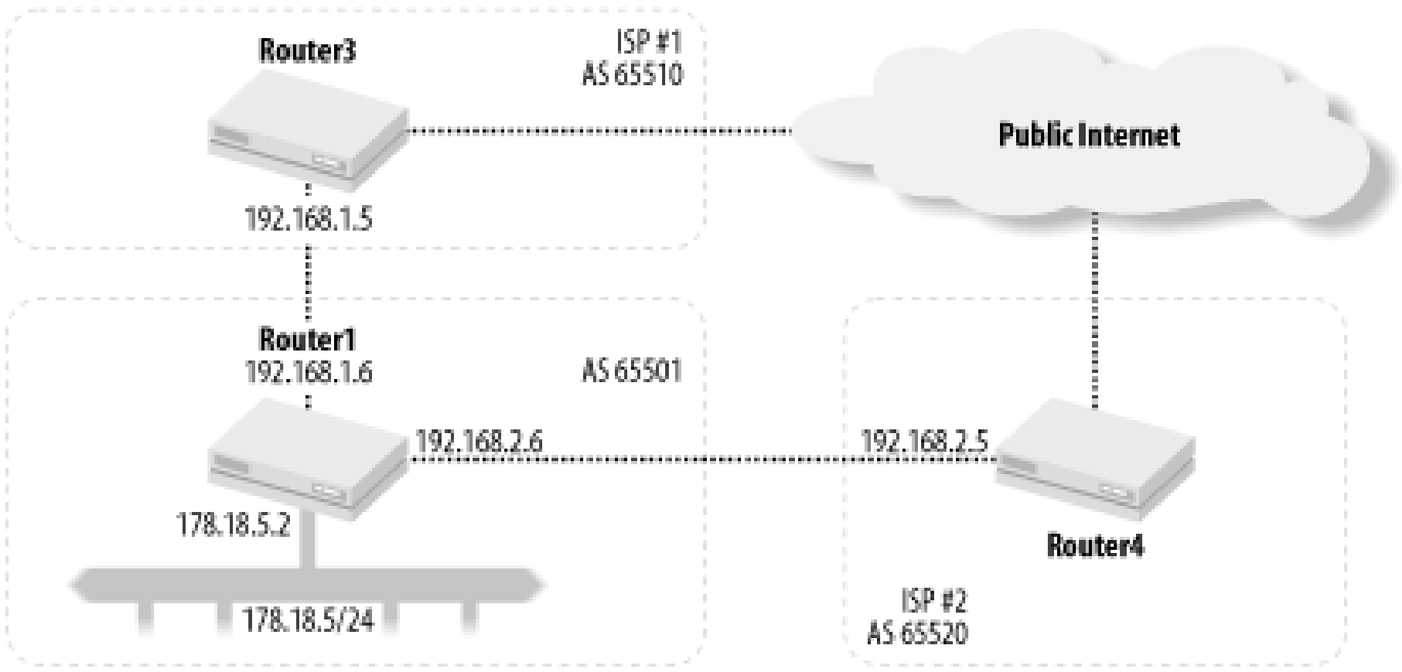
> Note that we do not recommend using this configuration as printed for a real Internet connection because it leaves out several key components. A more complete example is shown in Recipe 9.17.

9.4.3 Discussion

Perhaps the most common BGP application involves connecting a single router to two different ISPs to

share information about a single `/24` IP address range. A setup like this is the simplest way of building a redundant Internet connection. You can improve this redundancy by using two routers, one for each ISP connection, as shown in Recipe 9.5. Figure 9-1 shows the connections used in this recipe.

Figure 9-1. Using two ISPs



This example shows the configuration for the router at the customer site. The customer network uses ASN 65501, while the two ISPs use 65510 and 65520, respectively. Both of these connections are made through serial connections.

This configuration is a simple extension of the one shown in Recipe 9.1. The main difference is that we have set up two different peers, both in different ASes. This router is configured to distribute routing information for its `172.18.5.0/24` segment with both ISPs, and to receive their routing tables.

There are two critical problems with this simple configuration. First, the full Internet routing table is extremely large and consumes a vast amount of memory, so we will probably want to do some filtering. The second problem is that this configuration allows your network to act as a transit path between the two ISPs.

The full Internet routing table has well over 100,000 prefixes. Each BGP route entry consumes somewhere between 100 and 200 bytes of memory on the router, and you wouldn't use BGP unless there were at least two ISPs. Each ISP will likely supply a similar sized routing table, doubling the memory requirement. Then, if the router puts all of these prefixes into its main routing table (as well as in the CEF table), you can wind up consuming as much as 1KB of router memory per route prefix. So we don't recommend using a router with less than 100MB of memory when connecting to the Internet without significant filtering. In fact, Internet backbone routers frequently have hundreds of megabytes of memory.

Here is a typical routing summary taken from a BGP route server:

```
route-server.x>show ip route summary
Route Source    Networks    Subnets     Overhead    Memory (bytes)
connected       0           23          56          3680
```

```
static          1           4           280         2840
bgp xxx         87075       37990       7003640     20206240
  External: 0 Internal: 125065 Local: 0
internal        2078                                2452040
Total           89154       38017       7003976     22664800
route-server.x>
```

As you can see here, this router's routing table consumes most of the over 22MB of system memory. The same device uses roughly 20MB of memory just for its BGP table, as you can see from the following output:

```
route-server.x>show ip bgp summary
BGP router identifier xxx.xxx.xxx.xxx, local AS number xxx
BGP table version is 205557022, main routing table version 205557022
126231 network entries and 252452 paths using 20827755 bytes of memory

40380 BGP path attribute entries using 2099760 bytes of memory
173 BGP rrinfo entries using 4824 bytes of memory
24797 BGP AS-PATH entries using 606020 bytes of memory
336 BGP community entries using 16266 bytes of memory
Dampening enabled. 0 history paths, 0 dampened paths
BGP activity 2543115/2416883 prefixes, 11282422/11029970 paths

<information deleted for brevity>
route-server.x>
```

We discuss BGP route servers in more detail in Recipe 9.17.

Fixing the transit problem is somewhat easier than the route filtering that is necessary to reduce the size of the Internet route tables. To prevent the external networks from using your network for transit, you simply have to ensure that you never pass BGP routing information that you learn from one ISP over to the other ISP. This way neither ISP will know that it can reach the other through your network, so they won't send their traffic this way.

The easiest way to accomplish this is to put a filter on the AS path. In the following example, we will apply the same filter to both BGP peers. This filter will force our router to advertise only local routes. Any route that already has an entry in its AS path must have come from somewhere else, so we prevent the router from forwarding these routes. The router will add its own ASN to the AS path only after doing this filter processing, so the local routes will still be sent out:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#ip as-path access-list 15 permit ^$
Router1(config)#router bgp 65501
Router1(config-router)#network 172.18.5.0 mask 255.255.255.0
Router1(config-router)#neighbor 192.168.1.5 remote-as 65510
Router1(config-router)#neighbor 192.168.1.5 filter-list 15 out
Router1(config-router)#neighbor 192.168.2.5 remote-as 65520
Router1(config-router)#neighbor 192.168.2.5 filter-list 15 out
```

```
Router1(config-router)#end
Router1#
```

Please refer to Recipe 9.10 for more information on how to use AS filters.

Before you can solve the problem of the large size of the Internet routing tables, you have to make some decisions about how you want your Internet connections to work. Specifically, you might want one of these ISPs to be the primary and the other the backup for all traffic. Alternatively, you might want to just use the first ISP to handle traffic for its directly connected customers, while the second ISP handles everything else. Or you could opt to have load sharing between the two ISPs. These options are discussed in Recipe 9.7, Recipe 9.8, and Recipe 9.17.

You should also think about whether you want to control which path inbound traffic uses to reach you. If one of your ISP links has a large usage charge, you might prefer to force all of the inbound traffic through the other link. This can be slightly tricky, because you don't directly control the ISP routers. But you can control how your routing information looks to the ISP. Techniques for doing this are discussed in Recipe 9.13 and Recipe 9.17.

9.4.4 See Also

Recipe 9.5; Recipe 9.7; Recipe 9.8; Recipe 9.10; Recipe 9.13; Recipe 9.17

Top

Recipe 9.5 Connecting to Two ISPs with Redundant Routers

9.5.1 Problem

You want to connect your network to two different ISPs using two routers to eliminate any single points of failure.

9.5.2 Solution

In this example we have two routers in our AS, which has an ASN of 65500. The first router has a link to the first ISP, whose ASN is 65510:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#interface Serial0
Router1(config-if)#description connection to ISP #1, ASN 65510
Router1(config-if)#ip address 192.168.1.6 255.255.255.252
Router1(config-if)#exit
Router1(config)#interface Ethernet0
Router1(config-if)#description connection to internal network, ASN 65501
Router1(config-if)#ip address 172.18.5.2 255.255.255.0
Router1(config-if)#exit
Router1(config)#ip as-path access-list 15 permit ^$
Router1(config)#router bgp 65501
Router1(config-router)#network 172.18.5.0 mask 255.255.255.0
Router1(config-router)#neighbor 172.18.5.3 remote-as 65501
Router1(config-router)#neighbor 172.18.5.3 next-hop-self
Router1(config-router)#neighbor 192.168.1.5 remote-as 65510
Router1(config-router)#neighbor 192.168.1.5 filter-list 15 out
Router1(config-router)#no synchronization
Router1(config-router)#end
Router1#
```

The second router connects to the second ISP, which uses ASN 65520. And, because these two routers are both members of the same AS, they also must have an iBGP connection:

```
Router2#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router2(config)#interface Serial1
Router2(config-if)#description connection to ISP #2, ASN 65520
Router2(config-if)#ip address 192.168.2.6 255.255.255.252
Router2(config-if)#exit
Router2(config)#interface Ethernet0
```

```
Router2(config-if)#description connection to internal network, ASN 65501
Router2(config-if)#ip address 172.18.5.3 255.255.255.0
Router2(config-if)#exit
Router2(config)#ip as-path access-list 15 permit ^$
Router2(config)#router bgp 65501
Router2(config-router)#network 172.18.5.0 mask 255.255.255.0
Router2(config-router)#neighbor 192.168.2.5 remote-as 65520
Router2(config-router)#neighbor 192.168.2.5 filter-list 15 out
Router2(config-router)#neighbor 172.18.5.2 remote-as 65501
Router2(config-router)#neighbor 172.18.5.2 next-hop-self
Router2(config-router)#no synchronization
Router2(config-router)#end
Router2#
```

9.5.3 Discussion

This recipe is similar to Recipe 9.4, but here we have split the functions across two routers to ensure that you can sustain a link failure or a router failure without losing your Internet connection. Figure 9-2 shows the new network topology.

Figure 9-2. Using two ISPs with redundant routers

The main difference is that we have had to configure an eBGP link from each router to its ISP, as well as an iBGP link between the two routers. Note that we have included the same AS Path filter on both routers to ensure that our network doesn't allow transit routing from one ISP to the other.

However, just as in the single router example, you must decide how you want to deal with the problem of the excessive number of routes that you will receive from both of these ISPs.

Notice that we have included the *next-hop-self* option for the iBGP peers on both routers:

```
Router1(config)#router bgp 65501
Router1(config-router)#neighbor 172.18.5.3 remote-as 65501
Router1(config-router)#neighbor 172.18.5.3 next-hop-self
```

Without this option, the next-hop IP address for prefixes learned through Router1 will be the ISP connected to Router1. But, even in this simple network, Router2 will not have a route to this next-hop address. We could also get around this problem by including static routes on both routers. We discuss the *next-hop-self* option in more detail in Recipe 9.3.

In this example, we have only two routers inside our AS. You could add more, using exactly the same configuration commands that we used here. However, you need to remember to create a full mesh of iBGP peer relationships between all of these routers. Every BGP router must have a neighbor statement connecting to every other BGP router in the same AS.

9.5.4 See Also

Recipe 9.3; Recipe 9.4

Top

Recipe 9.6 Restricting Networks Advertised to a BGP Peer

9.6.1 Problem

You want to restrict which routes your router advertises to another AS.

9.6.2 Solution

There are three ways to filter routes in BGP. The first one uses extended access lists and route maps:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#access-list 105 deny ip host 172.25.0.0 host 255.255.0.0
Router1(config)#access-list 105 permit ip any any
Router1(config)#route-map ACL-RT-FILTER permit 10
Router1(config-route-map)#match ip address 105
Router1(config-route-map)#exit
Router1(config)#route-map ACL-RT-FILTER deny 20
Router1(config-route-map)#exit
Router1(config)#router bgp 65500
Router1(config-router)#neighbor 192.168.1.5 remote-as 65510
Router1(config-router)#neighbor 192.168.1.5 route-map ACL-RT-FILTER in
Router1(config-router)#end
Router1#
```

The second method uses a the *distribute-list* command:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#access-list 106 deny ip host 172.25.0.0 host 255.255.0.0
Router1(config)#access-list 106 permit ip any any
Router1(config)#router bgp 65500
Router1(config-router)#neighbor 192.168.1.5 remote-as 65510
Router1(config-router)#neighbor 192.168.1.5 distribute-list 106 in
Router1(config-router)#end
Router1#
```

But the most common way to filter routes in BGP is to use *prefix lists*. The following example has an effect to the preceding ones:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#ip prefix-list PREFIX-FILTER seq 10 deny 172.25.0.0/16
```

```
Router1(config)#ip prefix-list PREFIX-FILTER seq 20 permit 0.0.0.0/0 le 32
Router1(config)#router bgp 65500
Router1(config-router)#neighbor 192.168.1.5 remote-as 65510
Router1(config-router)#neighbor 192.168.1.5 prefix-list PREFIX-FILTER in
Router1(config-router)#end
Router1#
```

9.6.3 Discussion

In all of these examples, the router will suppress the route `172.25.0.0/16` from its BGP route table if it is received from the `192.168.1.5` eBGP peer. The first example uses route maps, the second uses a distribute list, and the third uses prefix lists. Examples of route maps and access lists appear throughout this book, so they should already be somewhat familiar. You can use them for a variety of different applications such as adjusting route tags, Local Preference, and BGP weight values. Here we just use the route map to look at the incoming routes from a peer device and reject certain routes.

The access list in the first example uses a "deny" clause to suppress the unwanted route, and ends with a "permit" command to allow all other routes to pass normally:

```
Router1(config)#access-list 105 deny ip host 172.25.0.0 host 255.255.0.0
Router1(config)#access-list 105 permit ip any any
```

Then the route map uses this access list to define which routes will be permitted to pass:

```
Router1(config)#route-map ACL-RT-FILTER permit 10
Router1(config-route-map)#match ip address 105
```

Then, we have added an explicit deny all clause to the route map that simply rejects anything that the first clause hasn't matched:

```
Router1(config-route-map)#route-map ACL-RT-FILTER deny 20
Router1(config-route-map)#exit
```

Note that every route map ends with an implicit deny all clause, so this was not strictly necessary. But it does make our intentions more clear to the next person who reads this router configuration.

For the distribute list example, we have created a normal access list that specifies the routes that are to be included or excluded from the distribution. This is almost identical to the distribute list technique that we discussed for RIP and EIGRP in Chapter 6 and Chapter 7, respectively.

Note the rather odd construction of the extended access lists in both the route map and the distribute list examples. As we discuss in Chapter 19, the first address and wildcard pair usually refers to the source, and the second set refers to the destination. But, in this case, we are actually trying to match specific route prefixes (not source and destination addresses), so the meanings are somewhat different. When filtering routes with extended ACLs, the first address defines the prefix, while the second part of the ACL defines the length of the prefix. This particular ACL matches the prefix `172.25.0.0/16`:

```
Router1(config)#access-list 105 permit ip host 172.25.0.0 host 255.255.0.0
```

If we had wanted to match the prefix `172.25.0.0/24` instead, we could have used and ACL that looks like this:

```
Router1(config)#access-list 105 permit ip host 172.25.0.0 host 255.255.255.0
```

You can also use standard ACLs for route filtering, but the results can be a little strange. Suppose we had used this ACL instead of the one we discussed above:

```
Router1(config)#access-list 5 permit 172.25.0.0
```

This will match `172.25.0.0/16`. But it doesn't specify the length of the prefix. So it will also match, for example, `172.25.0.0/24` if it exists. But it doesn't include any of the other subnets of `172.25.0.0/16`, such as `172.25.1.0/24`. We don't recommend using standard ACLs for route filtering because of this strange behavior.

Because it is so easy to get confused when using ACLs for matching prefixes, most people prefer to use prefix lists instead.

Prefix lists provide another way of doing the same kind of filtering. But it is often considerably easier to create useful filters with prefix lists because they were designed specifically for this purpose. Look at the prefix list in the example:

```
Router1(config)#ip prefix-list PREFIX-FILTER seq 10 deny 172.25.0.0/16
Router1(config)#ip prefix-list PREFIX-FILTER seq 20 permit 0.0.0.0/0 le 32
```

The first line of this list rejects the prefix `172.25.0.0/16`. The second line explicitly allows all other prefixes. Note that there is a sequence number in each line, specified by the argument of the *seq* keyword. This provides a convenient way of either inserting or removing new lines in the middle of a prefix list, as well as at the beginning or the end.

We suggest that you space these numbers in steps of 10, as we have done here, so that you can easily add lines. If you used a smaller step size between sequence numbers, you might find that there isn't enough room to add new rules. When this happens, you will have to delete the entire set of rules and reenter the commands with new sequence numbers.

Prefix lists show their real power when you want to deal with subnets. For example, suppose what you actually wanted to do was reject all of the subnets of `172.25.0.0/16`, while allowing a single summary route for the entire network. You could do this with the following prefix list:

```
Router1(config)#ip prefix-list PRE-RTFILTER seq 10 deny 172.25.0.0/16 ge 17
Router1(config)#ip prefix-list PRE-RTFILTER seq 20 permit 0.0.0.0/0 le 32
```

The first line rejects any subnets of `172.25.0.0/16` that have a prefix length of 17 bits or longer. So this would include, for example, `172.25.15.8/30`, `172.25.100.0/24`, and `172.25.252.0/22`. But this rule does not suppress the summary route (`172.25.0.0/16`) itself because it has a prefix length of only

16 bits. The rule only rejects prefixes that are 17 or more bits long.

This also helps to clarify the meaning of the second line of the prefix list. This line looks at any subnets of `0.0.0.0/0`, which is the entire IPv4 address range, and matches anything with a prefix length of 32 bits or less, which is everything.

You can also combine the *ge* and *le* keywords to create useful lists. For a slightly artificial example, if you wanted to permit all routes with prefixes of 8 to 16 bits, but nothing longer and nothing shorter, you could use the following single-line prefix list:

```
Router1(config)#ip prefix-list CLASS-A-B permit 0.0.0.0/0 ge 8 le 16
```

This also shows that you can match on prefix length independently of the actual network number. Notice also that we have omitted the sequence number in this example. By default, the router will rewrite this command and store it with a sequence number of 5 as follows:

```
ip prefix-list CLASS-A-B seq 5 permit 0.0.0.0/0 ge 8 le 16
```

If we were to then add another line to this list, the router would automatically give it sequence number 10, always incrementing in steps of 5. But we recommend using explicit sequence numbers to ensure that things are in the order you expect.

Here is a similar example that selects only the subnets of `172.25.0.0/16` that are between 19 and 24 bits long:

```
Router1(config)#ip prefix-list BIG-SUBNETS permit 172.25.0.0/16 ge 19 le 24
```

Once you have created a prefix list, you need to apply it to a neighbor statement using the *prefix-list* keyword as follows:

```
Router1(config)#router bgp 65500
Router1(config-router)#neighbor 192.168.1.5 remote-as 65510
Router1(config-router)#neighbor 192.168.1.5 prefix-list PREFIX-FILTER in
```

You have to define the peer with a *neighbor remote-as* command before you can apply any special options such as prefix lists to it. Otherwise the router will simply reject the command.

Note the keyword *in* at the end of the command. As with route maps, you can assign a prefix list either inbound or outbound using the keywords in or out respectively at the end of the line.

9.6.4 See Also

Chapter 6; Chapter 7; Chapter 19

Top

Recipe 9.7 Adjusting Local Preference Values

9.7.1 Problem

You want to change the Local Preference values to control which routes you use.

9.7.2 Solution

There are two ways to adjust Local Preference values on a router. The first method changes the Local Preference values for every route distributed into iBGP from this router:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#router bgp 65500
Router1(config-router)#bgp default local-preference 200
Router1(config-router)#end
Router1#
```

The second method uses route maps to give finer granularity control over which routes get what Local Preference values:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#ip prefix-list LOW_LP_PREFIXES seq 10 permit 172.22.0.0/16
Router1(config)#route-map LOCALPREF permit 10
Router1(config-route-map)#match ip address prefix-list LOW_LP_PREFIXES
Router1(config-route-map)#set local-preference 50
Router1(config-route-map)#exit
Router1(config)#route-map LOCALPREF permit 20
Router1(config-route-map)#exit
Router1(config)#router bgp 65500
Router1(config-router)#neighbor 192.168.1.5 remote-as 65510
Router1(config-router)#neighbor 192.168.1.5 route-map LOCALPREF in
Router1(config-router)#end
Router1#
```

9.7.3 Discussion

When BGP routers within an AS exchange information about a particular route using iBGP, they include the Local Preference value. All of the routers in the AS are then able to use this value to decide how to weight this route versus other BGP routes to the same destination. BGP consults the Local

Preference value early in the route selection process, before even the AS Path attribute. This provides an extremely useful and flexible way of forcing particular routes to use particular paths. Routers do not include Local Preference information when exchanging routes through eBGP connections.

A common example would be if you had two connections to an external network and you wanted to ensure that one was the primary path and the other was a backup. Suppose further that one of the routers in the AS handles the primary path and a second router handles the secondary path. The first example shows how to globally increase the Local Preference values of all routes received by one of these routers:

```
Router1(config)#router bgp 65500
Router1(config-router)#bgp default local-preference 200
```

Now all of the external routes that this router handles will have a Local Preference value of 200. If you can reach a particular prefix through more than one path, the other routers in this AS will prefer to use the one with the highest Local Preference value. The default Local Preference is 100.

To see how this works in practice, we will once again use the network shown in Figure 9-2. The following output shows two paths to the network `10.0.0.0/8`. This router learned the first route through iBGP from another router. You can see from the `LocPrf` column that this route has the default Local Preference value of 100:

```
Router2#show ip bgp
BGP table version is 4, local router ID is 172.18.5.3
Status codes: s suppressed, d damped, h history, * valid, > best, i - internal
Origin codes: i - IGP, e - EGP, ? - incomplete

   Network          Next Hop            Metric LocPrf Weight Path
* i10.0.0.0         172.18.5.2                100      0 65510 65531 i
*>                  192.168.2.5                        0 65520 65531 i
Router2#
```

This router learned the second route through eBGP, so it doesn't have a Local Preference value. The router treats any missing Local Preference values the as if they had the default value of 100. So, all other things being equal, this router prefers to use the more direct route that it learned itself, which is indicated by the > character at the beginning of the line.

Now we will change the default Local Preference value on the other router to 200, using the *bgp default local-preference* command shown earlier:

```
Router2#show ip bgp
BGP table version is 4, local router ID is 172.18.5.3
Status codes: s suppressed, d damped, h history, * valid, > best, i - internal
Origin codes: i - IGP, e - EGP, ? - incomplete

   Network          Next Hop            Metric LocPrf Weight Path
*>i10.0.0.0         172.18.5.2                200      0 65510 65531 i
*                   192.168.2.5                        0 65520 65531 i
```

```
Router2#
```

The Local Preference value has changed to 200 for the iBGP route, so this router now prefers the iBGP route. You can see more detail by specifying the prefix with this command:

```
Router2#show ip bgp 10.0.0.0/8
BGP routing table entry for 10.0.0.0/8, version 4
Paths: (2 available, best #1, table Default-IP-Routing-Table)
  Advertised to non peer-group peers:
  192.168.2.5
  65510 65531
    172.18.5.2 from 172.18.5.2 (172.18.5.2)
      Origin IGP, localpref 200, valid, internal, best
  65520 65531
    192.168.2.5 from 192.168.2.5 (172.21.1.1)
     Origin IGP, localpref 100, valid, external

Router2#
```

This clearly shows that the router interprets the missing Local Preference value as 100 on this router. If we now change the default value on this router to 75, it will use this new value instead when the value is missing:

```
Router2#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router2(config)#router bgp 65500
Router2(config-router)#bgp default local-preference 75
Router2(config-router)#end
Router2#clear ip bgp *
<wait until the BGP peers reconnect>
Router2#show ip bgp 10.0.0.0/8
BGP routing table entry for 10.0.0.0/8, version 2
Paths: (2 available, best #2, table Default-IP-Routing-Table)
  Advertised to non peer-group peers:
  192.168.2.5
  65520 65531
    192.168.2.5 from 192.168.2.5 (172.21.1.1)
     Origin IGP, localpref 75, valid, external

  65510 65531
    172.18.5.2 from 172.18.5.2 (172.18.5.2)
      Origin IGP, localpref 200, valid, internal, best
Router2#
```

Note that we had to clear the BGP peers and wait until they reconnected for this change to take effect. By chance, the router displays the routes in the opposite order. There is no particular significance to this ordering.

You can also use route maps to define different Local Preference values for different individual routes with route maps. This gives you a finer granularity, and even allows you to manually balance the load

between these links by forcing some routes through one path and the rest through another path.

The second example shows how to use a route map to adjust Local Preference values:

```
Router1(config)#ip prefix-list LOW_LP_PREFIXES seq 10 permit 172.22.0.0/16
Router1(config)#route-map LOCALPREF permit 10
Router1(config-route-map)#match ip address prefix-list LOW_LP_PREFIXES
Router1(config-route-map)#set local-preference 50
Router1(config-route-map)#exit
Router1(config)#route-map LOCALPREF permit 20
Router1(config-route-map)#exit
```

Here we have defined a prefix list that just matches the prefix `172.22.0.0/16`. Whenever the route map called LOCALPREF sees a route that matches this prefix list, it sets the Local Preference value for this route to 50.

We have included an empty clause at the end of the route map, which simply passes all other routes unchanged. Every route map ends with an implicit deny all. If we didn't include this explicit permit all clause, the router would simply drop any prefixes that didn't match the first clause.

We then invoke this rule using the standard *route-map* option to the *neighbor* command:

```
Router1(config)#router bgp 65500
Router1(config-router)#neighbor 192.168.1.5 remote-as 65510
Router1(config-router)#neighbor 192.168.1.5 route-map LOCALPREF in
```

Note that we are applying this route map to all incoming routes received from this specific eBGP peer. This particular route now has a Local Preference value of 50:

```
Router1#show ip bgp 172.22.0.0/16
BGP routing table entry for 172.22.0.0/16, version 5
Paths: (2 available, best #2, table Default-IP-Routing-Table)
Flag: 0x208
  Advertised to non peer-group peers:
  192.168.1.5
  65510 65531
    192.168.1.5 from 192.168.1.5 (172.25.26.55)
     Origin IGP, localpref 50, valid, external

  65520 65531
    172.18.5.3 from 172.18.5.3 (172.18.5.3)
      Origin IGP, localpref 75, valid, internal, best
Router1#
```

Note that the other iBGP router (which we configured earlier) is still using the Local Preference value of 75.

Because this method used a route map, you can easily construct rules that would change the Local Preference values based on a large variety of different parameters. For example, you could match based

on the AS Path, which is a technique that we discuss in more detail in Recipe 9.10. You could do this to give a higher or lower Local Preference value based on whether the route passes through a particular remote AS:

```
Router1(config)#ip as-path access-list 17 permit _65531_
Router1(config)#route-map LOCALPREF permit 30
Router1(config-route-map)#match as-path 17
Router1(config-route-map)#set local-preference 75
Router1(config-route-map)#exit
```

9.7.4 See Also

Recipe 9.10

Top

Recipe 9.8 Load Balancing

9.8.1 Problem

You want to load balance traffic over two or more links, between two eBGP or iBGP neighbors.

9.8.2 Solution

Although BGP goes to great lengths to ensure that there is only one path for each route by default, Cisco routers also allow you to configure load balancing for equal- cost paths:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#router bgp 65500
Router1(config-router)#maximum-paths 4
Router1(config-router)#end
Router1#
```

9.8.3 Discussion

This option is useful when there are multiple paths to a particular adjacent AS. As you can see from the following BGP route table, there are three different options for these routes:

```
Router1#show ip bgp
BGP table version is 12, local router ID is 172.18.5.2
Status codes: s suppressed, d damped, h history, * valid, > best, i - internal
Origin codes: i - IGP, e - EGP, ? - incomplete

   Network          Next Hop            Metric LocPrf Weight Path
*  10.0.0.0         192.168.1.5                            0 65510 65520 i
*>                  192.168.2.5                  0         0 65520 i
*                   192.168.3.5                            0 65520 i
*  172.25.0.0       192.168.1.5                            0 65510 65520 i
*>                  192.168.2.5                  0         0 65520 i
*                   192.168.3.5                            0 65520 i
Router1#
```

Without the maximum path option, there is only one route for each of these destinations in the IP routing table:

```
Router1#show ip route bgp
     172.25.0.0/16 is variably subnetted, 2 subnets, 2 masks
```

```
B        172.25.0.0/16 [20/0] via 192.168.2.5, 00:06:58
B     10.0.0.0/8 [20/0] via 192.168.2.5, 00:06:58
```

We then increase the maximum path value to 4 from the default of 1:

```
Router1(config)#router bgp 65500
Router1(config-router)#maximum-paths 4
```

The router now installs two routes in the IP routing table for each prefix:

```
Router1#show ip route bgp
     172.25.0.0/16 is variably subnetted, 2 subnets, 2 masks
B       172.25.0.0/16 [20/0] via 192.168.2.5, 00:00:02
                     [20/0] via 192.168.3.5, 00:00:02
B     10.0.0.0/8 [20/0] via 192.168.2.5, 00:00:02
                [20/0] via 192.168.3.5, 00:00:02
```

Note that the router did not install all three of the available BGP routes. This is because the other routes, the ones that use `192.168.1.5` for the next-hop router, both have a longer AS Path. Similarly, if one of these routes had a smaller Local Preference value, it also would not be used. As we mentioned when discussing the BGP route selection rules in the introduction to this chapter, the router will only use the BGP multipath feature for routes that are equivalent after all tests up to and including the MED test.

Also note that for eBGP connections, this only balances the outbound traffic load. Incoming packets are subject to whatever routing policies your neighboring AS uses.

Top

Recipe 9.9 Removing Private ASNs from the AS Path

9.9.1 Problem

You want to prevent your internal private ASNs from reaching the public Internet.

9.9.2 Solution

When using unregistered ASNs you have to be careful that they don't propagate into the public Internet.

In this example, the router has a BGP connection to an ISP, which uses ASN 1. Our router uses ASN 2, and connects to another router with an unregistered ASN, 65500:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#interface Serial0
Router1(config-if)#description connection to ISP #1, ASN 1
Router1(config-if)#ip address 192.168.1.6 255.255.255.252
Router1(config-if)#exit
Router1(config)#interface Serial1
Router1(config-if)#description connection to private network, ASN 65500
Router1(config-if)#ip address 192.168.5.1 255.255.255.252
Router1(config-if)#exit
Router1(config)#router bgp 2
Router1(config-router)#neighbor 192.168.5.2 remote-as 65500
Router1(config-router)#neighbor 192.168.1.5 remote-as 1
Router1(config-router)#neighbor 192.168.1.5 remove-private-AS
Router1(config-router)#no synchronization
Router1(config-router)#end
Router1#
```

9.9.3 Discussion

An unregistered ASN is a little bit like an unregistered IP address in that anybody can use it. So, if your routing prefixes have an unregistered ASN, and this information is eventually passed to another router that happens to be using the same unregistered ASN somewhere else in the Internet, that router will assume that there is a routing loop, and drop your routes.

Having said this, if you look on an Internet backbone router at any given moment, there is a reasonably good chance of seeing several unregistered ASNs being propagated. This is a dangerous situation, because the misbehaving networks could well be working perfectly today. But tomorrow, somebody

else could start using the same unregistered ASN. Every route from the first network will look like a loop when received by the second network. Two ASes will not be able to communicate if they both use the same ASN.

All of the work in this example is done by the simple *remove-private-AS* command. Here is what the BGP route table looks like on this router:

```
Router1#show ip bgp
BGP table version is 6, local router ID is 192.168.55.1
Status codes: s suppressed, d damped, h history, * valid, > best, i - internal
Origin codes: i - IGP, e - EGP, ? - incomplete

   Network          Next Hop            Metric LocPrf Weight Path
*> 10.0.0.0         172.20.1.2               0             0 1 i
*> 172.21.0.0       172.25.1.7               0             0 65500 i
*> 172.25.1.0/24    172.25.1.7               0             0 65500 i
Router1#
```

As you can see, we are receiving information about network `10.0.0.0` from the ISP router in AS 1, and `172.21.0.0` from the router with ASN 65500. Looking at the routes on the ISP router before turning on the *remove-private-AS* feature, you can see that this private ASN is propagating into the Internet, which is not allowed:

```
Router3# show ip bgp
BGP table version is 8, local router ID is 172.20.100.1
Status codes: s suppressed, d damped, h history, * valid, > best, i - internal
Origin codes: i - IGP, e - EGP, ? - incomplete

   Network          Next Hop            Metric LocPrf Weight Path
*> 10.0.0.0         0.0.0.0                  0         32768 i
*> 172.21.0.0       172.20.1.1                             0 2 65500 i
*> 172.25.1.0/24    172.20.1.1                             0 2 65500 i
Router3#
```

But after using the remove-private-AS command, all of the private ASNs are removed:

```
Router3# show ip bgp
BGP table version is 8, local router ID is 172.20.100.1
Status codes: s suppressed, d damped, h history, * valid, > best, i - internal
Origin codes: i - IGP, e - EGP, ? - incomplete

   Network          Next Hop            Metric LocPrf Weight Path
*> 10.0.0.0         0.0.0.0                  0         32768   i
*> 172.21.0.0       172.20.1.1                             0 2 i
*> 172.25.1.0/24    172.20.1.1                             0 2 i
Router3#
```

Be careful of this feature, though, because it can't remove private ASNs from the middle of an AS Path. If you have a topology where there is a public ASN behind a private one, it's not safe to remove the

private ASN because you could cause routing loops. So the *remove-private-AS* feature completely gives up and passes on the entire path for routes that have a public ASN after a private ASN.

If this is the case, your only recourse is to suppress the route with the illegal path. Then, as long as you distribute a prefix that includes this route, everything will work.

Top

Recipe 9.10 Filtering BGP Routes Based on AS Paths

9.10.1 Problem

You want to filter the BGP routes that you either send or receive based on AS Path information.

9.10.2 Solution

You can use AS Path filters either inbound or outbound, to filter either the routes you send or the routes you receive, respectively. You must apply these filters to each peer separately:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#ip as-path access-list 15 permit ^65501$
Router1(config)#ip as-path access-list 25 permit _65530_
Router1(config)#ip as-path access-list 25 deny _65531$
Router1(config)#ip as-path access-list 25 permit .*
Router1(config)#router bgp 65500
Router1(config-router)#neighbor 192.168.1.5 remote-as 65510
Router1(config-router)#neighbor 192.168.1.5 filter-list 15 in
Router1(config-router)#neighbor 192.168.2.5 remote-as 65520
Router1(config-router)#neighbor 192.168.2.5 filter-list 25 out
Router1(config-router)#end
Router1#
```

9.10.3 Discussion

One of the most common reasons for filtering routes based on the AS Path is to prevent AS transit as we showed in Recipe 9.4 and Recipe 9.5. However, there are some other useful applications for AS Path filters. The previous example contains two distinct filters, one of which is applied to routes received inbound from one neighbor, and the other works on outbound routes sent to a second neighbor.

AS Path filters are constructed using a subset of Unix regular expressions. Regular expressions provide an extremely powerful and general pattern-matching syntax. Many scripting languages such as Perl, Java, awk, sed, PHP, and Python use regular expressions for string manipulation. A detailed description of the syntax is out of the scope of this book, but fortunately, BGP path filters don't require all of the magic of the regular expression syntax. This is because all AS Paths consist of simply numbers separated by whitespace. There are no other characters to worry about, and every AS Path has a similar construction. Only the specific ASNs and the number of whitespaces ever change. For more

information on regular expressions in general, please refer to Mastering Regular Expressions (O'Reilly).

In Recipe 9.4, we showed a simple AS Path filter, that used the pattern "^$". In a regular expression, the symbol "^" stands for the start of the field, and "$" for the end. So the pattern "^$" simply means that the field is empty. In the case of a BGP AS Path, that means that this route must originate inside this AS.

Looking at the example above, then, it should be clear that access-list 15 looks for paths that contain only one ASN, which must be 65501:

```
Router1(config)#ip as-path access-list 15 permit ^65501$
```

Because there is both a "^" and a "$" in the pattern, this filter will match routes whose AS Path consists of just a single ASN, 65501. This filter will remove any downstream routes that AS 65501 is merely passing along. Also, as with normal access lists, AS Path filters end with an implicit deny all clause. The router will suppress any other routes that don't match this pattern.

The second AS Path filter in the example is somewhat more complicated:

```
Router1(config)#ip as-path access-list 25 permit _65530_
Router1(config)#ip as-path access-list 25 deny _65531$
Router1(config)#ip as-path access-list 25 permit .*
```

This shows that you can have filters that span multiple lines, although the example itself is a little bit artificial. The first line in this filter permits any routes that pass through AS 65530. The ASN in this line is surrounded by a "_" character. This character stands for whitespace, although it is a little bit confusing because, for example, `_65530_` seems to imply that it will match the ASN 65530 only if it appears in the middle of an AS Path. But, in fact, `_65530_` will match any path containing the ASN 65530, even if it is at the beginning or the end of the path. Conversely, `_65531$` will match only the AS Paths of those routes that originate in AS 65531.

This little _ delimiter character is extremely important because AS Path filters use a literal-text pattern-matching. For example, consider the following filter, which doesn't include this character:

```
Router1(config)#ip as-path access-list 26 permit 55
```

This AS Path filter will match not only paths containing AS 55, but any other ASN that happens to contain the digits 55, such as 65530, 7553 or 255. It is unlikely that you actually want to match on substrings within an ASN like this, so you should always remember to include these delimiter characters.

We included the following line in the recipe example because we needed to counteract the implicit deny all at the end of any AS Path access list:

```
Router1(config)#ip as-path access-list 25 permit .*
```

This statement explicitly permits all other AS Paths that have not matched any of the earlier lines in the filter rule. The character "." in this filter matches any character, while the "*" indicates that there can be any number of characters. In fact, "*" literally means zero or more matches. In cases where you actually need to match one or more times, you can use the "+" character.

There are many interesting uses for AS Path filters. For example, you might want to allow routes from an ISP and its immediate customers, but not from anything further away. This is easily accomplished with the following filter:

```
Router1(config)#ip as-path access-list 27 permit ^[0-9]+$
Router1(config)#ip as-path access-list 27 permit ^[0-9]+_[0-9]+$
```

This filter uses a couple of little tricks. The first trick is to specify a range, as in [0-9]. This means that the rule will match any character that falls in the range from 0 to 9, inclusive. Following this with the "+" character means that the rule matches one or more of these patterns. So, the first line in this filter matches all paths that contain one and only one ASN, although it doesn't matter what this ASN actually is. The second line similarly matches all paths that contain exactly two ASNs. The net effect is to allow only routes from the directly attached ISP AS, and from any other AS that is directly connected to the ISP.

Another way to write the same thing is to match on the delimiters in the AS Path, instead of the actual ASN values. To do this, you might use a pattern such as this:

```
Router1(config)#ip as-path access-list 28 deny _.+_.+_.+_
Router1(config)#ip as-path access-list 28 permit .*
```

In the first line of this access list, the "." character matches anything, including delimiters and digits. So this pattern will match an AS Path that includes at least four AS Path delimiters with something in between them. Since the first and last delimiters could be the beginning and end of the AS Path, rather than actual whitespace, this access list causes the router to suppress any AS Path that includes three or more ASNs. It's slightly confusing because you have to think in terms of matching on delimiters rather than ASNs, but the net effect of AS Path access list number 28 is identical to 27. And, if you wanted to increase the maximum number of ASN values in the path from 2 to, say, 5, this syntax is much more flexible:

```
Router1(config)#ip as-path access-list 29 deny _.+_.+_.+_.+_.+_.+_
Router1(config)#ip as-path access-list 29 permit .*
```

It's useful to remember that you can affect not only the routes you receive, but also the routes that you send using AS Path filters. In Recipe 9.4, we showed an extremely useful technique that uses AS Path filters to prevent an AS from being used for transit between external networks:

```
Router1(config)#ip as-path access-list 15 permit ^$
Router1(config)#router bgp 65500
Router1(config-router)#neighbor 192.168.1.5 remote-as 65520
Router1(config-router)#neighbor 192.168.1.5 filter-list 15 out
```

In this case, the filter permits only routes that have an empty AS Path, meaning that the routes must have originated locally within this AS. This filter suppresses any external routing information when forwarding its routing table. So the external networks don't know about any downstream networks that can be reached through this router.

You could use a slightly more complicated outbound filter if you wanted. This example allows only directly connected networks to use your AS for transit:

```
Router1(config)#ip as-path access-list 16 deny _.+_.+_
Router1(config)#ip as-path access-list 16 permit .*
Router1(config)#router bgp 65500
Router1(config-router)#neighbor 192.168.1.5 remote-as 65520
Router1(config-router)#neighbor 192.168.1.5 filter-list 16 out
```

The router applies this filter before it adds itself to the AS Path. So, when we deny the pattern `_.+_.+_`, this suppresses all AS Paths with two or more ASNs, leaving only AS Paths that have a single ASN. Any path with one ASN must originate in a directly connected AS.

This AS Path filter might seem a little bit confusing because it denies paths that we don't want rather than permitting the ones we do. If you prefer, you could create a filter that has the identical effect, by explicitly permitting only the paths that we want:

```
Router1(config)#ip as-path access-list 17 permit ^[0-9]+$
Router1(config)#ip as-path access-list 17 permit ^$
```

Both of these filters allow the router to forward routing information that originates in this AS, and in any networks that are directly connected to us. Bear in mind that this doesn't prevent a device that is 15 hops away from reaching one of our neighbors through our network. But it does prevent them from reaching anything more distant than one of our direct neighbors through our AS.

9.10.4 See Also

Recipe 9.4; Recipe 9.5; Mastering Regular Expressions, by Jeffrey E.F. Friedl (O'Reilly)

Top

Recipe 9.11 Reducing the Size of the Received Routing Table

9.11.1 Problem

You want to summarize the incoming routing information to reduce the size of your routing table.

9.11.2 Solution

One of the easiest ways to reduce your routing table size is to filter out most of the external routes and replace them with a default. To do this, first create a static default route pointing to some known remote network. If this remote network is up, you can safely assume that your ISP is working properly. Then you simply filter out all of the remaining uninteresting routes:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#ip route 0.0.0.0 0.0.0.0  192.168.101.0 1
Router1(config)#ip route 0.0.0.0 0.0.0.0  192.168.102.0 2
Router1(config)#ip prefix-list CREATE-DEFAULT seq 10 permit 192.168.101.0/24
Router1(config)#ip prefix-list CREATE-DEFAULT seq 20 permit 192.168.102.0/24
Router1(config)#router bgp 65500
Router1(config-router)#neighbor 192.168.1.5 remote-as 65520
Router1(config-router)#neighbor 192.168.1.5 prefix-list CREATE-DEFAULT in
Router1(config-router)#end
Router1#
```

9.11.3 Discussion

For most typical Internet connections, you will need to drastically reduce the amount of routing information that you receive. A typical Internet backbone router needs to support BGP routes for well over 100,000 prefixes. So, unless you are the ISP and need to support a large fraction of the public address space, it is a good idea to cut out as much as possible. It is important to remember that removing routing information means that some of your routing decisions will not be as good as they might otherwise be, however. There is always a tradeoff involved in filtering routing information.

This recipe shows a good way to drastically reduce the size of your Internet routing table. It looks for two different remote networks on the Internet, `192.168.101.0/24` and `192.168.102.0/24` , and points a default route to each of them. This way, if either route happens to fail because of some normal (but hopefully rare) network problem, you will still have a default route. Then we created a prefix list that allows only these two routes, and applied it to all routes that we received from the peer router at our ISP. Please refer to Recipe 9.6for more information on prefix lists.

The result is a very small Internet routing table that consists of only these two routes and a default

route. In practice, you will probably want to use more than two routes, however. Just to guard against the possibility that the remote networks you picked happen to be down at the same time for some reason. It is a good idea to pick a wide variety of different remote networks, some very far away and some relatively close. Avoid picking all of them in the same country, so you won't lose your default just because of a telecom disaster in that country. You could even pick a dozen or so remote routes like this, giving excellent fault tolerance, while still providing a tiny Internet routing table.

Note that the two static routes in the example have different administrative distances:

```
Router1(config)#ip route 0.0.0.0 0.0.0.0  192.168.101.0 1
Router1(config)#ip route 0.0.0.0 0.0.0.0  192.168.102.0 2
```

We did this to prevent load balancing between the default routes. If you have more than one ISP, it is quite likely that the best routes for these prefixes will be through different providers. You can allow load balancing, if you prefer, by simply giving all of these static routes the same administrative distance and including the *maximum-paths* command (as we discussed in Recipe 9.8). But bear in mind that this will balance among routes, not among ISP connections.

If you have only one ISP, load balancing between these default routes accomplishes nothing useful.

If you then want to pass this default route information along to other routers using BGP, the best way to do so is to use the *default-originate* option on the *neighbor* command, and include a route map to specify the prefixes that you want to associate with your default route:

```
Router1(config)#ip prefix-list CREATE-DEFAULT seq 10 permit 192.168.101.0/24
Router1(config)#ip prefix-list CREATE-DEFAULT seq 20 permit 192.168.102.0/24
Router1(config)#route-map DEFAULT-ROUTE permit 10
Router1(config-route-map)#match ip address prefix-list CREATE-DEFAULT
Router1(config-route-map)#exit
Router1(config)#router bgp 65500
Router1(config-router)#neighbor 172.18.5.3 default-originate route-map DEFAULT-ROUTE
Router1(config-router)#exit
```

This is a dangerous thing to do, though, because BGP will now start to distribute default routing information to this peer, which may then start to distribute the default route out to the Internet. It is a good idea to explicitly suppress the default route on all of your BGP routers for any peers that should not receive it:

```
Router1(config)#ip prefix-list BLOCK-DEFAULT permit 0.0.0.0/0 ge 1
Router1(config)#router bgp 65500
Router1(config-router)#neighbor 192.168.1.5 prefix-list BLOCK-DEFAULT out
```

Another popular way to reduce the size of the Internet routing table is to simply refuse to accept any route's `/24` prefixes. Over 50% of the routes appearing on the Internet backbone are for `/24` prefixes, so eliminating these will cut the memory requirements in half:

```
Router1(config)#ip prefix-list BLOCK-24 permit 0.0.0.0/0 le 23
```

```
Router1(config)#router bgp 65500
Router1(config-router)#neighbor 192.168.1.5 prefix-list BLOCK-24 in
```

However, if you do this, you should also use a default static route method discussed earlier. This is because many of the `/24` prefixes in the Internet routing tables are not included in other prefixes or summary routes.

The fraction of routes appearing on the backbone with a `/24` prefix is steadily dropping over time. In early 2001, almost 59% of all prefixes were`/24` networks, while over two years later, the number had dropped to roughly 55%. We expect this trend to continue over time, as ISPs improve their route summarization.

9.11.4 See Also

Recipe 9.4 ; Recipe 9.5 ; Recipe 9.6 ; Recipe 9.8

Top

Recipe 9.12 Summarizing Outbound Routing Information

9.12.1 Problem

You want to summarize your routing table before forwarding it to another router.

9.12.2 Solution

BGP includes an automatic summarization feature that is on by default:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#router bgp 65501
Router1(config-router)#neighbor 172.21.2.1 remote-as 65530
Router1(config-router)#auto-summary
Router1(config-router)#end
Router1#
```

9.12.3 Discussion

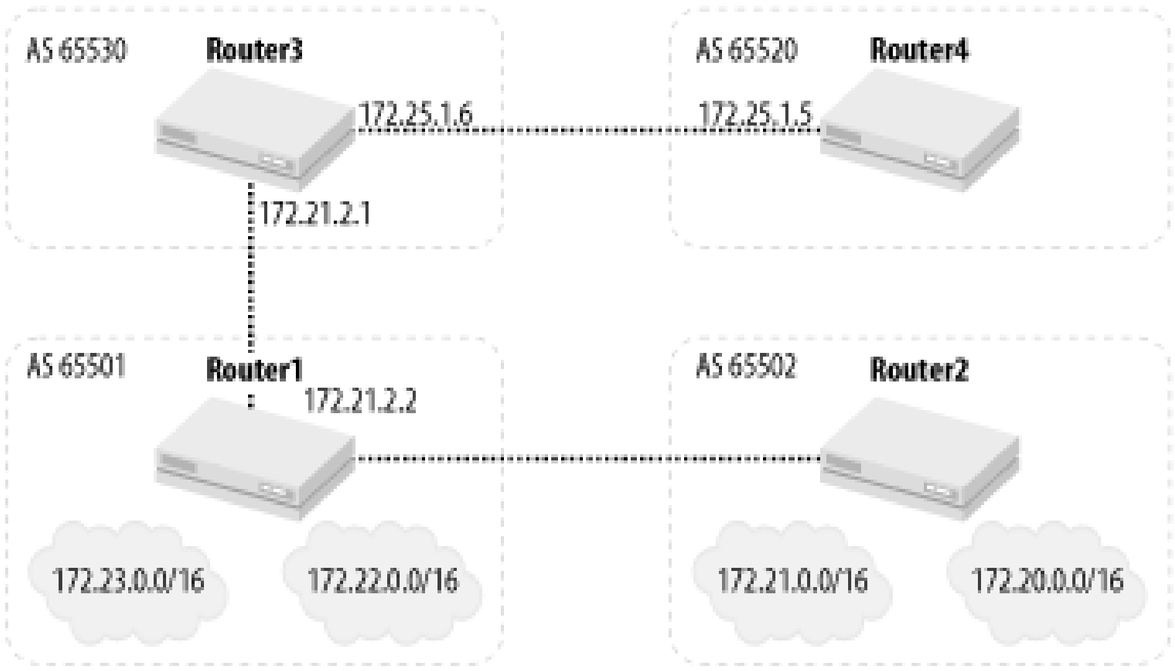
By default, BGP will try to summarize routes. This is not always desirable, though, which is why we have explicitly disabled this feature in many of the examples in this chapter. In fact, many engineers prefer to manually summarize their routing tables because they want to control what gets summarized and what doesn't.

The first problem with auto-summarization is that it is strictly classful. Your AS may not control all of the subnets in a classful network, and even if you do, this may not be a useful prefix to summarize your networks. The second problem is that it only works on routes that are redistributed into BGP, and not on those received from other BGP routers or injected via the *network* command. Please refer to Recipe 9.14 for more information on redistributing routes into BGP.

Suppose you wanted to summarize several routes to a single non-classful route, or to summarize routes from several downstream BGP networks. You might be tempted to handle this by redistributing a static route for the summary and suppressing the individual routes with a filter. The problem with doing this is that the static route never goes away, even if all of the routes that you are trying to summarize become unreachable.

Cisco gets around this problem by implementing a special *aggregate-address* command that allows you to do the summarization without needing to manually create some routes and suppress others. Figure 9-3 shows a network using route aggregation.

Figure 9-3. Route aggregation



Suppose the engineer responsible for AS 65530 wants to summarize the routes he receives from AS 65501 before passing this information along to another AS such as AS 65520. Router1 in AS 65501 advertises the prefixes `172.20.0.0/16` and `172.21.0.0/16` , which it learned from Router2 in AS 65502, and adds to it the prefixes`172.22.0.0/16` and `172.23.0.0/16` . All of these networks are covered by the aggregate address, `172.20.0.0/14` :

```
Router3(config)#router bgp 65530
Router3(config-router)#aggregate-address 172.20.0.0 255.252.0.0  summary-only
```

The *summary-only* keyword here means that BGP will suppress the individual subnets. On the router doing the route aggregation you can see which routes will be suppressed:

```
Router3#show ip bgp
BGP table version is 29, local router ID is 172.20.100.1
Status codes: s suppressed, d damped, h history, * valid, > best, i - internal
Origin codes: i - IGP, e - EGP, ? - incomplete

   Network          Next Hop            Metric LocPrf Weight Path
<routes deleted for brevity>
s> 172.20.0.0       172.21.2.2               0             0 65501 65502 ?
*> 172.20.0.0/14    0.0.0.0                              32768 i
s> 172.21.0.0       172.21.2.2               0             0 65501 65502 ?
s> 172.21.0.0       172.21.2.2               0             0 65501 ?
s> 172.22.0.0       172.21.2.2               0             0 65501 ?
s> 172.23.0.0       172.21.2.2               0             0 65501 ?
```

Then, in downstream ASes, such as AS 65520, there is no indication of the summarized networks:

```
Router4#show ip bgp
BGP table version is 284, local router ID is 172.27.9.1
Status codes: s suppressed, d damped, h history, * valid, > best, i - internal
Origin codes: i - IGP, e - EGP, ? - incomplete

   Network          Next Hop        Metric LocPrf Weight Path
<routes deleted for brevity>
```

```
*> 172.20.0.0/14   172.25.1.6                          0 65530 i
Router4#
```

If you omit the *summary-only* keyword, BGP will advertise the summary address as well as the summarized subnets:

```
Router4#show ip bgp
BGP table version is 284, local router ID is 172.27.9.1
Status codes: s suppressed, d damped, h history, * valid, > best, i - internal
Origin codes: i - IGP, e - EGP, ? - incomplete

   Network          Next Hop        Metric LocPrf Weight Path
<routes deleted for brevity>
*> 172.20.0.0      172.25.1.6                           0 65530 65501 65502 ?
*> 172.20.0.0/14   172.25.1.6                           0 65530 i
*> 172.21.0.0      172.25.1.6                           0 65530 65501 65502 ?
*> 172.22.0.0      172.25.1.6                           0 65530 65501 ?
*> 172.23.0.0      172.25.1.6                           0 65530 65501 ?
Router4#
```

As long as the router doing the aggregation continues to see any routes that are within the summarized range, it will advertise the summary route. However, if all of the component routes disappear, it will stop advertising the summary. This is true whether you use the *summary-only* keyword or not:

```
Router4#show ip bgp 172.20.0.0
% Network not in table
```

Route summarization inherently discards information. To see why this can cause problems, suppose there was a link between Router1 and Router4. Router4 will advertise the summary route, which does not have Router1's ASN in the AS Path. So Router1 will accept this as a new, distinct route that passes through Router4. If Router1 then loses its route to one of the summarized addresses, say `172.23.0.0/16`, it will try to use the summary route, and send packets for this prefix to Router4. Router4 will forward the packets to Router3. If Router3 still has the suppressed route in its BGP table, it will simply forward the packet back to Router1, completing a routing loop.

Router3 will eventually purge the unreachable prefix from its routing table, but in more complex networks, it could take a while for this to happen.

To get around this problem, BGP includes the concept of an AS Set that can be used with route aggregation. An AS Set is a grouping of ASNs in an AS Path. It indicates that the route passed through one or more of the listed ASes. Because the AS Path now contains every ASN, you can again eliminate loops.

You can enable AS Sets with the *as-set* keyword in the *aggregate-address* command:

```
Router3(config)#router bgp 65530
Router3(config-router)#aggregate-address 172.20.0.0 255.255.252.0  as-set summary-only
```

Then, on a downstream router, the *show ip bgp* output includes the AS Set and represents it in curly braces:

```
Router4#show ip bgp
BGP table version is 36, local router ID is 172.25.26.5
Status codes: s suppressed, d damped, h history, * valid, > best, i - internal
Origin codes: i - IGP, e - EGP, ? - incomplete

   Network          Next Hop       Metric LocPrf Weight Path
<routes deleted for brevity>
*> 172.20.0.0/14   172.25.1.6                           0 65530 {65501,65502} ?
Router4#
```

You can see more detail by specifying the route prefix with the *show ip bgp* command. Note that this output even tells you the BGP router ID of the router that did the aggregation, as well as the ASN that this router resides in:

```
Router4#show ip bgp 172.20.0.0
BGP routing table entry for 172.20.0.0/14, version 36
Paths: (1 available, best #1, table Default-IP-Routing-Table)
  Not advertised to any peer
  65530 {65501,65502}, (aggregated by 65530 172.20.100.1)
    172.25.1.6 from 172.25.1.6 (172.27.9.1)
      Origin incomplete, localpref 100, valid, external, best
Router4#
```

You need to be careful with route summarization, particularly when you don't control all of the subnets in the range that you intend to summarize. In our example, suppose we advertised the summary for `172.20.0.0/14` but we didn't know how to route some part of this range, such as `172.21.15.0/24`.

In an ideal world, this wouldn't actually matter because the real owner of `172.21.15.0/24` and its subnets would advertise a more precise route than our summary. But this is not a completely ideal world, and sometimes people might filter out the longer masks as a matter of course to reduce their routing tables (as we did in Recipe 9.11). It is entirely possible that our router will be called upon to route packets for a device in `172.21.15.0/24`. If our response to this is simply to toss the packet back to our default gateway, we could easily wind up with a routing loop.

If you intend to summarize, make sure that you can vouch for all of the subnets that you are summarizing. This is true regardless of what techniques you use.

9.12.4 See Also

Recipe 9.11 ; Recipe 9.14

Top

Recipe 9.13 Prepending ASNs to the AS Path

9.13.1 Problem

You want to increase the length of an AS Path so that one inbound path looks better than another.

9.13.2 Solution

In situations where you have multiple connections between ASes, you will often want to make remote networks prefer one inbound path when sending packets to your network. The easiest way to do this is to prepend your own ASN to the AS Path several times, instead of just once as it would do by default:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#ip as-path access-list 15 permit ^$
Router1(config)#route-map PREPEND permit 10
Router1(config-route-map)#match as-path 15
Router1(config-route-map)#set as-path prepend 65501 65501 65501
Router1(config-route-map)#route-map PREPEND permit 20
Router1(config-route-map)#exit
Router1(config)#router bgp 65501
Router1(config-router)#neighbor 192.168.1.5 remote-as 65510
Router1(config-router)#neighbor 192.168.1.5 route-map PREPEND out
Router1(config-router)#end
Router1#
```

This example uses the same network shown in Figure 9-2.

9.13.3 Discussion

We have already discussed methods for making your outbound traffic prefer one path over another in Recipe 9.7. If you also want to ensure that inbound traffic prefers one path over another, you have to somehow trick the remote networks into believing that one path is better than the other.

As we mentioned in the introduction to this chapter, if there are many options for different paths to a destination network, a BGP router will go through several steps to decide which one to use. You can adjust the attributes associated with each route to help force other BGP routers to select the paths that you want them to use. The easiest way to force routers outside of your AS to favor a particular route is to adjust the AS Path.

If you can simply make the path appear longer for routes that use one link, then remote networks will tend to prefer to reach you through whatever other links are available. There will always be situations where it is still closer to use the route with the artificially lengthened path. But these situations should be relatively rare, and the more times you prepend your ASN to the path, the more rare they will be.

Of course, it isn't safe or wise to put an arbitrary ASN into the AS Path. But you can insert your own ASN a few extra times without causing any problems, which is exactly what this recipe shows. Note that there is no hard limit to how long your AS Path can be (although it would probably cause problems if the path were so long that the routing information couldn't fit into a single BGP packet), and some sites prepend their ASN 10 or 20 times to make absolutely certain that a particular path is used only in case of a failure of the primary path. However, the longest AS Paths in the public Internet rarely have more than a dozen ASNs. You shouldn't need to prepend your ASN very many times to make one path look better than the other from anywhere in the Internet.

This recipe also takes the precaution of only lengthening the AS Paths of locally generated routes. It does this by including a *match* clause in the route map that only affects routes that have an empty AS Path. Clause number 20 in the route map is a catchall that simply passes through all other routes unchanged:

```
Router1(config)#ip as-path access-list 15 permit ^$
Router1(config)#route-map PREPEND permit 10
Router1(config-route-map)#match as-path 15
Router1(config-route-map)#set as-path prepend 65501 65501 65501
Router1(config-route-map)#route-map PREPEND permit 20
Router1(config-route-map)#exit
```

But you might not want this restriction. You might prefer to rewrite all of the routes that you send. Or, you might use an outbound filter, such as the one discussed in Recipe 9.4, to completely suppress external routes. In both of these cases, you can make the route map considerably simpler:

```
Router1(config)#route-map PREPEND permit 10
Router1(config-route-map)#set as-path prepend 65501 65501 65501
Router1(config-route-map)#route-map PREPEND permit 20
Router1(config-route-map)#exit
```

The difference caused by prepending your ASN to the AS Path of a route is only visible on a remote router:

```
Router3#show ip bgp 172.18.5.0/24
BGP routing table entry for 172.18.5.0/24, version 26
Paths: (2 available, best #2)
  Advertised to non peer-group peers:
    192.168.1.6
  65501 65501 65501 65501
     192.168.1.6 from 192.168.1.6 (172.18.5.2)
       Origin IGP, metric 0, localpref 100, valid, external, ref 2
  65531 65520 65501
```

```
    192.168.99.6 from 192.168.99.6 (192.168.99.10)
      Origin IGP, localpref 100, valid, external, best, ref 2
Router3#
```

Here you can see that there are two routes for the prefix `172.18.5.0/24`: one passes through AS 65501, and the other passes through ASes 65531 and 65520 to reach AS65501. The path that goes directly to AS 65501 is actually shorter. But because we have prepended the ASN three times on this route, this router prefers the other path.

You can also verify that everything is working properly by disabling the peer relationship with the preferred ISP and making sure that everything still works. You can temporarily disable a peer by using the *shutdown* keyword on the *neighbor* command:

```
Router1(config)#router bgp 65501
Router1(config-router)#neighbor 192.168.2.5 shutown
```

Make sure to reenable this peer after you have finished testing:

```
Router1(config)#router bgp 65501
Router1(config-router)#no neighbor 192.168.2.5 shutown
```

9.13.4 See Also

Recipe 9.4; Recipe 9.7

Top

Recipe 9.14 Redistributing Routes with BGP

9.14.1 Problem

You want to redistribute routes between an IGP and BGP.

9.14.2 Solution

When connecting two or more IGPs together using BGP, you sometimes need to configure redistribution between the IGP and BGP on both routers. To make the example more interesting, we will assume that we need to connect an EIGRP network to an OSPF network using a pair of BGP routers.

The first router redistributes routes from BGP into OSPF:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#router ospf 100
Router1(config-router)#network 172.26.0.0 0.0.255.255  area
Router1(config-router)#redistribute bgp 65500 metric 500 subnets
Router1(config-router)#exit
Router1(config)#router bgp 65500
Router1(config-router)#neighbor 192.168.1.5 remote-as 65520
Router1(config-router)#network 172.26.0.0
Router1(config-router)#end
Router1#
```

This is the configuration for the router that redistributes BGP routes into EIGRP:

```
Router2#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router2(config)#router eigrp 99
Router2(config-router)#network 172.25.0.0
Router2(config-router)#redistribute bgp 65520 metric 500 10 255 1 1500
Router2(config-router)#exit
Router2(config)#router bgp 65520
Router2(config-router)#neighbor 192.168.1.6 remote-as 65500
Router2(config-router)#network 172.25.0.0
Router2(config-router)#end
Router2#
```

9.14.3 Discussion

Before we say anything about this recipe, we need to stress that redistribution with BGP is often a very messy business. The example specifically shows using BGP to handle routing between two IGPs rather than redistribution into the public Internet, because this technique is relevant only in a large enterprise network. Even here, it's easy to create routing loops, particularly when redistributing from BGP into an IGP and then back out into BGP. This is why we have actually chosen to distribute from BGP into the other protocols, rather than full two-way redistribution. We will discuss two-way redistribution in a moment.

For Internet connections, we strongly recommend against redistributing routes from BGP into the IGP. This is because it is too easy to inadvertently wind up distributing over 100,000 Internet routing prefixes into your IGP. And, when passing IGP routing information to the Internet, it is better to pass a few summary routes using *network* statements, as we have done in the previous example, than to directly redistribute IGP prefixes.

In addition, there is a huge danger when you redistribute BGP routes from the Internet into an IGP, back into BGP, and then onto the Internet. Unless you carefully filter routes (or your ISP filters for you), you will wind up sending routes back into the Internet with very short AS Paths that originate in your network. This could reroute the entire Internet through your IGP, which would be a bad thing.

However, BGP isn't just used on the public Internet. Many large enterprise networks also use BGP for interconnecting IGPs. In this case, although you have to redistribute routes from BGP into the IGP, it may not be necessary to redistribute IGP routes into BGP. So, in the above example, we have actually only used one-way redistribution from the BGP into the IGP, but not the other way around. Instead, we have relied on network statements to summarize the IGP routes into BGP:

```
Router1(config)#router bgp 65500
Router1(config-router)#neighbor 192.168.1.5 remote-as 65520
Router1(config-router)#network 172.26.0.0
```

If you need BGP to advertise a large number of IGP routes, you can use as many *network* statements as necessary. Note that before IOS Version 12.0, you could configure a maximum of 200 *network* statements. However, Cisco has now removed this restriction.

Having said all of this, it is possible to redistribute prefixes from the IGP into BGP. Two-way redistribution can be convenient when you want to use BGP to connect two IGPs that use overlapping address ranges. This might happen, for example, if a single IGP became too large and needed to be split for stability reasons.

We have already discussed how to redistribute foreign routing protocols into both EIGRP and OSPF in Chapter 7 and Chapter 8, so we will focus instead on the redistribution of these protocols into BGP.

The work is all done by a simple *redistribute* command in the BGP configuration clause. For simple redistribution, all we need to do is specify the IGP protocol and its process ID number:

```
Router1(config)#router bgp 65500
Router1(config-router)#redistribute ospf 100
```

B default, when you do this BGP takes the IGP metric and uses it as the BGP metric, also called the MED. BGP distributes this metric with the route. Routers use the lowest metric value to select the best route if two or more BGP routes have the same AS Path length.

This selects the BGP router that is closest to the IGP destination network, which may be exactly the right behavior. But, in some cases, you might want to force a specific metric value to ensure that a particular BGP router or link is used when routing to IGP destinations. You can do this most easily by simply setting a default metric for all redistributed IGP routes:

```
Router2(config)#router bgp 65520
Router2(config-router)#redistribute eigrp 99 metric 500
```

Although we have assigned a default metric only to the EIGRP routes, you can use the same syntax to give a default metric to the routes redistributed from OSPF.

We also mentioned in Chapter 7 and Chapter 8 that you can assign a route tag to external routes in both OSPF and EIGRP. This starts to become useful when you do lots of redistribution. In particular, suppose that the EIGRP and OSPF sides of this network have a back door connection to one another, redistributing part or all of their routing tables. In this case, we will need to restrict which routes we redistribute into BGP and suppress all of these back door routes or we will have routing loop problems.

You can do this with a route map. The simplest case, which is useful for the back door example, is to suppress all external routes when redistributing from the IGP into BGP:

```
Router2(config)#route-map REDIST deny 10
Router2(config-route-map)#match route-type external
Router2(config-route-map)#exit
Router2(config)#route-map REDIST permit 20
Router2(config-route-map)#exit
Router2(config)#router bgp 65520
Router2(config-router)#redistribute eigrp 99 route-map REDIST metric 500
```

You would need to apply a similar route map at both redistribution points to prevent loops.

However, the route map we just created will match all external routes, not just the external routes due to the back door connection. You might have some other redistributed static routes in your IGP, or perhaps an isolated part of your network uses RIP to support legacy equipment. This is where route tags become invaluable. If you have external routes that you need to redistribute into BGP, and you know that they have a particular tag value such as 123, you can use a slightly more complicated route map:

```
Router2(config)#route-map REDIST permit 5
Router2(config-route-map)#match tag 123
Router2(config-route-map)#exit
Router2(config)#route-map REDIST deny 10
Router2(config-route-map)#match route-type external
```

```
Router2(config-route-map)#exit
Router2(config)#route-map REDIST permit 20
Router2(config-route-map)#exit
Router2(config)#router bgp 65520
Router2(config-router)#redistribute eigrp 99 route-map REDIST metric 500
```

This route map now allows BGP to redistribute all local routes and all external routes that have the tag value 123, but it suppresses all other external routes. You might use a redistribution system like this if you were using BGP to act as a transit between two networks. The router that redistributes routes into BGP from the other network would mark them with this tag value. Then, at this router, we can use this tag to select only these particular external routes for distribution into this particular IGP.

9.14.4 See Also

Chapter 7; Chapter 8

Top

Recipe 9.15 Using Peer Groups

9.15.1 Problem

You want to apply the same options to several peers.

9.15.2 Solution

Peer groups allow you to apply the same BGP configuration to a number of neighbors at the same time:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#router bgp 65500
Router1(config-router)#neighbor EBGP-PEERS peer-group
Router1(config-router)#neighbor EBGP-PEERS prefix-list PRE-RTFILTER in
Router1(config-router)#neighbor EBGP-PEERS filter-list 15 out
Router1(config-router)#neighbor 192.168.1.5 remote-as 65520
Router1(config-router)#neighbor 192.168.1.5 peer-group EBGP-PEERS
Router1(config-router)#neighbor 192.168.1.9 remote-as 65521
Router1(config-router)#neighbor 192.168.1.9 peer-group EBGP-PEERS
Router1(config-router)#neighbor 192.168.1.13 remote-as 65522
Router1(config-router)#neighbor 192.168.1.13 peer-group EBGP-PEERS
Router1(config-router)#neighbor 192.168.1.17 remote-as 65523
Router1(config-router)#neighbor 192.168.1.17 peer-group EBGP-PEERS
Router1(config-router)#end
Router1#
```

9.15.3 Discussion

Peer groups have been around since IOS Version 11.0, but they had several unfortunate restrictions that were eliminated in Version 12.0. The most important of these were that all eBGP members of the same peer group had to be members of the same IP subnet and routers couldn't perform transit functions for eBGP neighbors that were members of the same peer group. These restrictions have been removed, but you will still sometimes see these problems discussed in older references.

Peer groups are most useful when you have several neighbors, all with nearly the same BGP parameters. In the previous example, we created a peer group called *EBGP-PEERS* that we can then applied to several different neighbors. This allows you to set up common properties such as filter lists or route maps, and apply them identically to a large list of peers.

The biggest value for this feature is for ISPs who want to set up common properties for all of the other ISP routers at a large Internet exchange point. But peer groups can also be useful in enterprise networks that include several ASes that all connect with one another.

Suppose, for example, that you need to connect to four different ASes, and that each connection point used two BGP routers for redundancy. Each router would have an eBGP connection to two routers in each of three different ASes, for a total of six eBGP connections.

If you need to do any special filtering on one of these routers, then you would have to configure six different eBGP neighbors with an identical set of filters. Or, if you use peer groups, you can set up the filters just once and reduce the typing as well as the chance of errors. Further, if you need to make any changes to your filters, you can make them once and they will instantly apply to all of the group members.

It's also important to remember that you can use the peer group as a basic template, but still add further options for one or more individual peers:

```
Router1(config)#router bgp 65500
Router1(config-router)#neighbor EBGP-PEERS peer-group
Router1(config-router)#neighbor EBGP-PEERS prefix-list PRE-RTFILTER in
Router1(config-router)#neighbor EBGP-PEERS filter-list 15 out
Router1(config-router)#neighbor 192.168.1.5 remote-as 65520
Router1(config-router)#neighbor 192.168.1.5 peer-group EBGP-PEERS
Router1(config-router)#neighbor 192.168.1.5 ebgp-multihop 5
```

Top

Recipe 9.16 Authenticating BGP Peers

9.16.1 Problem

You want to authenticate your BGP peer relationships to help prevent tampering with your routing tables.

9.16.2 Solution

The BGP protocol includes an MD5-based authentication system for authenticating peers:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#router bgp 65500
Router1(config-router)#neighbor 192.168.55.5 remote-as 65501
Router1(config-router)#neighbor 192.168.55.5 password password-1234
Router1(config-router)#end
Router1#
```

The same password must be configured on both routers:

```
Router2#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router2(config)#router bgp 65501
Router2(config-router)#neighbor 192.168.55.6 remote-as 65500
Router2(config-router)#neighbor 192.168.55.6 password password-1234
Router2(config-router)#end
Router2#
```

9.16.3 Discussion

MD5 authentication is a standard part of BGP Version 4 that was introduced in RFC 2385. The IETF went further in RFC 3013 (which is also called BCP 46) to recommend that "BGP authentication should be used with routing peers" in the public Internet. This language, "should be used," indicates a strong recommendation, but not a requirement.

BGP is different from the routing protocols that we discussed in Chapter 6, Chapter 7, and Chapter 8 because you must explicitly configure the peer relationships between routers. These peers then use point-to-point TCP connections to exchange information, which makes it much more difficult for a malicious user to surreptitiously establish a peer relationship with one of your routers and corrupt your

routing tables. But it is still possible to hijack an existing TCP connection between two BGP peers and inject bad routes. And, if the attacker is on the same network segment as one of the peers, they can potentially hijack the IP address of the legitimate peer and set up a new BGP session.

With authentication, this type of attack is considerably more difficult. This is because the attacker must not only get the TCP sequence numbers right, but he must also insert the correct encrypted authentication key.

Some sources have claimed that this MD5 authentication scheme is not sufficient for BGP because there are effective attacks that can break it. The Internet Draft document, "Security Requirements for Keys Used with the TCP MD5 Signature Option" (http://www.ietf.org/internet-drafts/draft-ietf-idr-md5-keys-00.txt) comments on this threat and makes the following recommendations:

- Make your keys between 12 and 24 bytes long.
- In situations with multiple BGP peers, avoid using the same keys with all peers.
- Change your keys at least every 90 days.

It is also important to note that introducing authentication can cause delays in BGP message passing, although it shouldn't seriously affect normal IP packet processing. It can also cause increased CPU overhead on the BGP peer routers.

Despite all of this, in a hostile network, authentication can be useful because it makes it significantly harder for somebody to disrupt your routing tables. If your ISP supports this service, it is probably a good idea to use it.

It is also worth mentioning that, in your router's configuration file, the password will be stored in plain-text unless you have enabled the *service password-encryption* global configuration command. When you turn on password encryption, the router will store the command using the Cisco proprietary type 7 encryption:

```
!
router bgp 65500
neighbor 192.168.55.5 remote-as 65501
neighbor 192.168.55.5 password 7 15020A1F173D24362C7E64704053
!
```

As we mentioned in Chapter 3, it is quite easy to break this encryption algorithm. But as long as you maintain good control over your router's configuration files, this will at least prevent somebody from learning the encryption key by looking over your shoulder.

When there is an authentication mismatch between two BGP peers, they will not be able to establish a connection. You will also see the following error message on one or both routers:

```
Jan  7 10:01:48 EST: %TCP-6-BADAUTH: No MD5 digest from 192.168.55.6:13662 to 192.
168.55.5:179
```

9.16.4 See Also

Chapter 3; RFC 3013; "Security Requirements for Keys Used with the TCP MD5 Signature Option", http://www.ietf.org/internet-drafts/draft-ietf-idr-md5-keys-00.txt, February 2002

Top

Recipe 9.17 Putting It All Together

9.17.1 Problem

You want to combine the best elements of this chapter to create a good redundant ISP connection.

9.17.2 Solution

For simplicity, we will extend the single router/dual ISP configuration of Recipe 9.4 , rather than using the dual router/dual ISP example of Recipe 9.5. It should be clear from the discussion in Recipe 9.5 how to extend this example to the two router case:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#interface Serial0
Router1(config-if)#description connection to ISP #1, ASN 65510
Router1(config-if)#ip address 192.168.1.6 255.255.255.252
Router1(config-if)#exit
Router1(config)#interface Serial1
Router1(config-if)#description connection to ISP #2, ASN 65520
Router1(config-if)#ip address 192.168.2.6 255.255.255.252
Router1(config-if)#exit
Router1(config)#interface Ethernet0
Router1(config-if)#description connection to internal network, ASN 65501
Router1(config-if)#ip address 172.18.5.2 255.255.255.0
Router1(config-if)#exit
Router1(config)#ip as-path access-list 15 permit ^$
Router1(config)#ip route 0.0.0.0 0.0.0.0 192.168.101.0 1
Router1(config)#ip route 0.0.0.0 0.0.0.0 192.168.102.0 2
Router1(config)#ip prefix-list CREATE-DEFAULT seq 10 permit 192.168.101.0/24
Router1(config)#ip prefix-list CREATE-DEFAULT seq 20 permit 192.168.102.0/24
Router1(config)#ip prefix-list BLOCK-DEFAULT seq 10 permit 0.0.0.0/0 ge 1
Router1(config)#route-map PREPEND permit 10
Router1(config-route-map)#set as-path prepend 65501 65501
Router1(config-route-map)#exit
Router1(config)#route-map LOCALPREF permit 10
Router1(config-route-map)#set local-preference 75
Router1(config-route-map)#exit
Router1(config)#route-map DEFAULT-ROUTE permit 10
Router1(config-route-map)#match ip address prefix-list CREATE-DEFAULT
Router1(config-route-map)#exit
Router1(config)#router bgp 65501
Router1(config-router)#network 172.18.5.0 mask 255.255.255.0
Router1(config-router)#neighbor 172.18.5.3 remote-as 65501
Router1(config-router)#neighbor 172.18.5.3 password password_number1
Router1(config-router)#neighbor 172.18.5.3 default-orignate route-map DEFAULT-ROUTE
```

```
Router1(config-router)#neighbor 192.168.1.5 remote-as 65510
Router1(config-router)#neighbor 192.168.1.5 password password_number2
Router1(config-router)#neighbor 192.168.1.5 filter-list 15 out
Router1(config-router)#neighbor 192.168.1.5 prefix-list CREATE-DEFAULT in
Router1(config-router)#neighbor 192.168.1.5 prefix-list BLOCK-DEFAULT out
Router1(config-router)#neighbor 192.168.2.5 remote-as 65520
Router1(config-router)#neighbor 192.168.2.5 password password_number3
Router1(config-router)#neighbor 192.168.2.5 filter-list 15 out
Router1(config-router)#neighbor 192.168.2.5 prefix-list CREATE-DEFAULT in
Router1(config-router)#neighbor 192.168.2.5 prefix-list BLOCK-DEFAULT out
Router1(config-router)#neighbor 192.168.2.5 route-map PREPEND out
Router1(config-router)#neighbor 192.168.2.5 route-map LOCALPREF in
Router1(config-router)#no synchronization
Router1(config-router)#end
Router1#
```

9.17.3 Discussion

In this recipe we put together several of the concepts discussed throughout the chapter. This router has three BGP peers, two of which are ISPs, while the other is an internal BGP router.

We have disabled synchronization. We aren't using an IGP on this router, so synchronization doesn't serve any purpose. We have used a *network* statement that covers only part of a classful network:

```
Router1(config)#router bgp 65501
Router1(config-router)#network 172.18.5.0 mask 255.255.255.0
Router1(config-router)#no synchronization
```

All of the peer relationships, including the internal peer, use MD5 authentication, which we have configured using the *neighbor password* command as discussed in Recipe 9.16:

```
Router1(config)#router bgp 65501
Router1(config-router)#neighbor 172.18.5.3 password password_number1
Router1(config-router)#neighbor 192.168.1.5 password password_number2
Router1(config-router)#neighbor 192.168.2.5 password password_number3
```

Note that we have configured different passwords on each peer, and each password is between 12 and 24 characters long, as discussed in Recipe 9.16.

We have configured an AS Path filter to each of the ISP peers to prevent them from using our network for transit purposes:

```
Router1(config)#router bgp 65501
Router1(config)#ip as-path access-list 15 permit ^$
Router1(config-router)#neighbor 192.168.1.5 filter-list 15 out
Router1(config-router)#neighbor 192.168.2.5 filter-list 15 out
```

We have followed Recipe 9.11 to replace the entire Internet routing table with a default route. This router then passes its default route along to the internal BGP router, which forces us to be careful that

we don't distribute the default back to the ISP routers:

```
Router1(config)#ip route 0.0.0.0 0.0.0.0 192.168.101.0 1
Router1(config)#ip route 0.0.0.0 0.0.0.0 192.168.102.0 2
Router1(config)#ip prefix-list CREATE-DEFAULT seq 10 permit 192.168.101.0/24
Router1(config)#ip prefix-list CREATE-DEFAULT seq 20 permit 192.168.102.0/24
Router1(config)#ip prefix-list BLOCK-DEFAULT permit 0.0.0.0/0 ge 1
Router1(config)#route-map DEFAULT-ROUTE permit 10
Router1(config-route-map)#match ip address prefix-list CREATE-DEFAULT
Router1(config-route-map)#exit
Router1(config)#router bgp 65501
Router1(config-router)#neighbor 172.18.5.3 remote-as 65501
Router1(config-router)#neighbor 172.18.5.3 default-origniate route-map DEFAULT-ROUTE
Router1(config-router)#neighbor 192.168.1.5 remote-as 65510
Router1(config-router)#neighbor 192.168.1.5 prefix-list CREATE-DEFAULT in
Router1(config-router)#neighbor 192.168.1.5 prefix-list BLOCK-DEFAULT out
Router1(config-router)#neighbor 192.168.2.5 remote-as 65520
Router1(config-router)#neighbor 192.168.2.5 prefix-list CREATE-DEFAULT in
Router1(config-router)#neighbor 192.168.2.5 prefix-list BLOCK-DEFAULT out
```

Next, we have used Recipe 9.7 to make ISP #2 less attractive for outbound traffic. This may be because this ISP has higher usage charges, or perhaps it is a lower bandwidth connection. Using Local Preference for this ensures that all of the BGP routers inside the AS can share this information about the best outbound path:

```
Router1(config)#route-map LOCALPREF permit 10
Router1(config-route-map)#set local-preference 75
Router1(config-route-map)#exit
Router1(config)#router bgp 65501
Router1(config-router)#neighbor 192.168.2.5 route-map LOCALPREF in
```

And finally, we have followed Recipe 9.13to make sure that inbound traffic from the public Internet also prefers ISP #1:

```
Router1(config)#route-map PREPEND permit 10
Router1(config-route-map)#set as-path prepend 65501 65501
Router1(config-route-map)#exit
Router1(config)#router bgp 65501
Router1(config-router)#neighbor 192.168.2.5 route-map PREPEND out
```

You should feel free to mix and match these types of configuration elements to make your configuration match your requirements.

Many backbone ISPs have *looking glass* servers that allow you to see how your BGP routes look several hops away from your network. These are generally web pages that allow you to submit*show bgp* -type queries for specific routes. You can find a list of looking glass servers around the world on http://www.traceroute.org . This site also lists a large number of traceroute servers, which will allow you to test which paths inbound connections will use to reach your network.

9.17.4 See Also

Recipe 9.4 ; Recipe 9.5 ; Recipe 9.7 ; Recipe 9.13 ; Recipe 9.16

Top

Chapter 10. Frame Relay

Top

Introduction

Frame Relay is a popular WAN protocol because it makes it easy to construct reliable and inexpensive networks. Its main advantage over simple point-to-point serial links is the ability to connect one site to many remote sites through a single physical circuit. Frame Relay uses *virtual circuits* to connect any physical circuit in a cloud to any other physical circuit. Many virtual circuits can coexist on a single physical interface.

This section will offer only a quick refresher of how Frame Relay works. If you are unfamiliar with Frame Relay, we recommend reading the more detailed description of the protocol and its features that are found in T1: A Survival Guide (O'Reilly).

The Frame Relay standard allows for both Switched (SVC) and Permanent (PVC) Virtual Circuits, although support for SVCs in Frame Relay switching equipment continues to be relatively rare. Most fixed Frame Relay WANs use PVCs rather than SVCs. This allows you to configure the routers to look like a set of point-to-point physical connections. SVCs, on the other hand, provide a mechanism for the network to dynamically make connections between any two physical circuits as they are needed. In general, SVCs are more complicated to configure and manage. Most network engineers prefer to use PVCs unless the carrier offers significant cost benefits for using SVCs. SVCs tend to be most practical when the site-to-site traffic is relatively light and intermittent.

Each virtual circuit is identified by a Data Link Connection Identifier (DLCI), which is simply a number between 0 and 1023. In fact, Cisco routers can only use DLCI numbers in the range 16 through 1007 to carry user data.

If the router at Site A wants to send a packet to Site B, it simply specifies the appropriate DLCI number for the virtual circuit that connects to Site B in the Frame Relay header. Although a physical circuit can have many virtual circuits, each connecting to a different remote circuit, there is no ambiguity about where the network should send each individual packet.

It's important to remember, though, that the DLCI number only has local significance. That is, the DLCI number doesn't uniquely identify the whole virtual circuit, just the connection from the local physical circuit to the Frame Relay switch at the Telco central office. The DLCI number associated with this virtual circuit can change several times before it reaches the remote physical circuit.

We like to use this fact to our advantage when constructing a Frame Relay network. Instead of thinking of the DLCI number as a virtual circuit identifier, we use it to uniquely label each physical circuit. Suppose, for example, that Site A has virtual circuits to both Sites B and C. Then we would use the same DLCI number at both Sites B and C to label the virtual circuits that terminate at Site A. This is just one of many possible DLCI numbering schemes, but we prefer it because it makes troubleshooting

easier. Unfortunately, while this scheme works well in hub-and-spoke network topologies, it tends to become unworkable in meshed or partially meshed networks.

Frame Relay QoS Features

Frame Relay has several built-in Quality of Service (QoS) features. Each virtual circuit has two important service level parameters, the Committed Information Rate (CIR) and the Excess Information Rate (EIR). The CIR is the contracted minimum throughput of a virtual circuit. As long as you send data at a rate that is less than the CIR value, it should all arrive. The EIR is the available capacity above the CIR. The worst case is when the router is sending data through a single virtual circuit at the line speed of the physical circuit. The network will generally just drop all packets that exceed the EIR, so it is customary to have the sum of CIR and EIR for each virtual circuit equal the line speed of the physical circuit. This makes it physically impossible to exceed the EIR for any PVC.

When the router sends packets faster than the CIR rate that you have contracted with your network provider, the carrier network may drop some or all of the excess packets if there is congestion in the cloud. To indicate which packets are in the excess region, the first switch to receive them will often mark the Discard Eligible (DE) bit in their Frame Relay headers. If there is no congestion, the packet will be delivered normally. But, if the packet goes through a congested part of the carrier's network, the switches will know that they can drop this packet without violating the CIR commitments. By just counting the packets that have their DE bit set on the receiving router, you get a useful measure of how often your network exceeds the CIR on each PVC. Because your traffic patterns will probably not be symmetrical in most networks, you should monitor the number of DE packets received separately on both ends of every PVC.

By default, the router will send frames into the cloud without the DE bit set. What happens next is up to the carrier, but it is common for the first switch to monitor the incoming traffic rate using some variation of the following. During each sample period (typically a short period of time such as a second), the switch will count the incoming bytes on each PVC. If there is more data than the CIR for this PVC, the switch will mark the DE bit in all of the excess frames.

However, it is also possible to configure the router to set the DE bit on low priority traffic in the hopes that the network will drop these low priority packets in preference to the high priority packets. This is something of a gamble, of course, and its success depends critically on the precise algorithm that your WAN vendor uses for handling congestion. You should consult with your vendor to understand their traffic shaping and policing mechanisms before attempting this type of configuration.

There are two other extremely useful flags in the Frame Relay header. These are the Forward Explicit Congestion Notification (FECN) and the Backward Explicit Congestion Notification (BECN) bits. These simply indicate that the packet encountered congestion somewhere in the carrier's network. Congestion is most serious when you are sending at a rate higher than the CIR value; if your carrier marks the DE bit of these excess packets, then congestion in one of their switches could mean dropped

packets.

If a packet encounters congestion in a carrier switch, that switch will often set the FECN flag in the packet's header. Then, when the other router finally receives this packet, it will know that it was delayed. But this is actually not all that useful, because the receiving router is not able to directly affect the rate that the sending router forwards packets along this virtual circuit. So the Frame Relay standard also includes the BECN flag.

When a switch encounters congestion and needs to set the FECN flag on a packet, it looks for another packet traversing the same PVC in the opposite direction, and marks it with a BECN flag. This way, the sending router immediately knows that the packets it is sending are encountering congestion.

Note, however, that not all Frame Relay switches implement these features in the same way. So, just because you don't see any FECN or BECN frames doesn't mean you can safely assume that there is no congestion. Similarly, not seeing DE frames doesn't necessarily mean that you aren't exceeding the CIR for a PVC. In Recipe 10.6, for example, we show how to configure a router to act as a Frame Relay switch. But the router does not implement these congestion notification features at all. The DE counter is also not a meaningful indicator of how often you exceed CIR if your devices are configured to send low priority frames with the DE bit set.

By default, the router will not react to FECN and BECN markings. Recipe 10.9 shows how you can look at statistics on FECN and BECN frames to get an idea of the network performance. In the next chapter, Recipe 11.11 shows how to configure a router to automatically adapt to congestion in the carrier network by reading and responding to the BECN flags, reducing the sending rate until the congestion disappears.

Top

Recipe 10.1 Setting Up Frame Relay withPoint-to-Point Subinterfaces

10.1.1 Problem

You want to configure Frame Relay services so that every PVC is assigned to a separate subinterface.

10.1.2 Solution

Probably the cleanest way to set up a Frame Relay network is to use point-to-point subinterfaces. If you have a host site that connects to two or more branches through a Frame Relay WAN, you could configure the central host router like this:

```
Central#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Central(config)#interface Serial0
Central(config-if)#description Frame-Relay host circuit
Central(config-if)#no ip address
Central(config-if)#encapsulation frame-relay
Central(config-if)#exit
Central(config)#interface Serial0.1 point-to-point
Central(config-subif)#description PVC to first branch - DLCI 101
Central(config-subif)#ip address 192.168.1.5 255.255.255.252
Central(config-subif)#frame-relay interface-dlci 101
Central(config-fr-dlci)#exit
Central(config-subif)#exit
Central(config)#interface Serial0.2 point-to-point
Central(config-subif)#description PVC to second branch - DLCI 102
Central(config-subif)#ip address 192.168.1.9 255.255.255.252
Central(config-subif)#frame-relay interface-dlci 102
Central(config-fr-dlci)#end
Central#
```

And all of the branches would follow the same basic configuration, but with different IP addresses and DLCI numbers:

```
Branch1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Branch1(config)#interface Serial0
Branch1(config-if)#description Frame-Relay circuit
Branch1(config-if)#no ip address
Branch1(config-if)#encapsulation frame-relay
Branch1(config-if)#exit
Branch1(config)#interface Serial0.1 point-to-point
```

```
Branch1(config-subif)#description PVC to Central host - DLCI 50
Branch1(config-subif)#ip address 192.168.1.6 255.255.255.252
Branch1(config-subif)#frame-relay interface-dlci 50
Branch1(config-fr-dlci)#end
Branch1#
```

10.1.3 Discussion

In this example, we have assumed that all of the Frame Relay circuits connect to serial interfaces on the routers. This is normally the case, but there are other options. Frame Relay is usually delivered on low speed 56 or 64Kbps circuits, or fractional or full T1 or E1 circuits. However, there are useful Frame Relay implementations all the way up to T3 speeds. The most common way to deliver Frame Relay service faster than T1 or E1 speeds is on either a coax T3 or a High Speed Serial Interface (HSSI) connection.

In all cases, the router is the Data Terminal Equipment (DTE) device, and the Frame Relay switch in the carrier's network is the Data Communications Equipment (DCE). Make sure that you have the right type of cable on your router, with a DTE connector.

While many carriers currently offer T3 Frame Relay service, very few Frame Relay switches are able to reliably switch packets along a single PVC much faster than T1 or E1 speeds. This means that a T3 or HSSI circuit makes an excellent aggregation point for a large number of branches with T1, E1, or slower circuits. However, you should talk it over very thoroughly with your WAN provider before attempting to build a Frame Relay network that requires CIR rates greater than a T1 or E1.

By default, the router will use a Cisco proprietary encapsulation format for the data payload of each packet. If you have to connect to non-Cisco equipment, you may prefer to use the open standard encapsulation format described in RFC 1490 instead. You can configure this either for each subinterface separately, or globally for the entire interface. To configure one subinterface to use RFC 1490 encapsulation, use the *ietf* keyword:

```
Central(config)#interface Serial0.1 point-to-point
Central(config-subif)#frame-relay interface-dlci 101 ietf
Central(config-fr-dlci)#end
```

You can make RFC 1490 encapsulation the default for all subinterfaces on an interface as follows:

```
Central(config)#interface Serial0
Central(config-if)#encapsulation frame-relay ietf
Central(config-if)#end
```

When you do this, you do not need to specify the *ietf* keyword on each subinterface. However, if you want to use to the Cisco encapsulation format on a particular PVC, you can do so with the *cisco* keyword:

```
Central(config)#interface Serial0.1 point-to-point
```

```
Central(config-subif)#frame-relay interface-dlci 101 cisco
Central(config-fr-dlci)#end
```

It is extremely important to specify the *point-to-point* keyword here. The problem is that you can't change a subinterface type. If you specify the wrong type of subinterface, you must delete the incorrect one, then reboot the router before you can re-create it with the correct type. This was particularly serious in earlier IOS releases, because the default was *multipoint*, rather than *point-to-point*. In Version 12.0 and higher, there is no default, and you must explicitly specify either point-to-point or multipoint. We will discuss multipoint subinterfaces in Recipe 10.4.

The *show frame-relay pvc* command shows the status and several useful statistics for every PVC:

```
Central#show frame-relay pvc

PVC Statistics for interface Serial0 (Frame Relay DTE)

DLCI = 101, DLCI USAGE = LOCAL, PVC STATUS = ACTIVE, INTERFACE = Serial0.1

  input pkts 4092          output pkts 1331         in bytes 573274
  out bytes 364868         dropped pkts 0           in FECN pkts 0
  in BECN pkts 0           out FECN pkts 0          out BECN pkts 0
  in DE pkts 0             out DE pkts 0
  out bcast pkts 1277       out bcast bytes 361391
  pvc create time 21:16:46, last time pvc status changed 21:16:46

DLCI = 102, DLCI USAGE = LOCAL, PVC STATUS = DELETED, INTERFACE = Serial0.2

  input pkts 0             output pkts 2            in bytes 0
  out bytes 566            dropped pkts 0           in FECN pkts 0
  in BECN pkts 0           out FECN pkts 0          out BECN pkts 0
  in DE pkts 0             out DE pkts 0
  out bcast pkts 2          out bcast bytes 566
  pvc create time 00:02:08, last time pvc status changed 00:01:15
Central#
```

In this case, two DLCIs are configured on the router. Only one of these is in an active state, while the other shows as `DELETED`, which means that it is not configured on the switch. This command will also show you if there are other PVCs configured in the Frame Relay switch, but not on the router. These DLCIs are easy to spot because the `DLCI USAGE` field is listed as `UNUSED`:

```
Central#show frame-relay pvc

PVC Statistics for interface Serial1 (Frame Relay DTE)

DLCI = 101, DLCI USAGE = LOCAL, PVC STATUS = ACTIVE, INTERFACE = Serial0.1

  input pkts 11            output pkts 14           in bytes 2218
  out bytes 1825           dropped pkts 3           in FECN pkts 0
  in BECN pkts 0           out FECN pkts 0          out BECN pkts 0
  in DE pkts 0             out DE pkts 0
```

```
  out bcast pkts 9          out bcast bytes 1305
  pvc create time 00:02:45, last time pvc status changed 00:02:24

DLCI = 102, DLCI USAGE = LOCAL, PVC STATUS = DELETED, INTERFACE = Serial0.2

  input pkts 0              output pkts 2           in bytes 0
  out bytes 566             dropped pkts 0          in FECN pkts 0
  in BECN pkts 0            out FECN pkts 0         out BECN pkts 0
  in DE pkts 0              out DE pkts 0
  out bcast pkts 2          out bcast bytes 566
  pvc create time 00:02:08, last time pvc status changed 00:01:15

DLCI = 103, DLCI USAGE = UNUSED, PVC STATUS = INACTIVE, INTERFACE = Serial0

  input pkts 0              output pkts 0           in bytes 0
  out bytes 0               dropped pkts 0          in FECN pkts 0
  in BECN pkts 0            out FECN pkts 0         out BECN pkts 0
  in DE pkts 0              out DE pkts 0
  out bcast pkts 0          out bcast bytes 0           Num Pkts Switched 0
  pvc create time 00:00:08, last time pvc status changed 00:00:08

Central#
```

In this case, you can see that a new PVC with DLCI 103 was created on the switch eight seconds ago on the circuit that connects to the router's `Serial0` interface. This new PVC is not associated with a subinterface, and it is not passing any traffic.

The *show interface* command gives other useful information, particularly about the Local Management Interface (LMI) protocol:

```
Branch#show interface serial0
Serial0 is up, line protocol is up
  Hardware is HD64570
  Description: Frame-Relay circuit
  MTU 1500 bytes, BW 1544 Kbit, DLY 20000 usec, rely 255/255, load 1/255
  Encapsulation FRAME-RELAY, loopback not set, keepalive set (10 sec)
  LMI enq sent  7932, LMI stat recvd 7932, LMI upd recvd 0, DTE LMI up
  LMI enq recvd 0, LMI stat sent  0, LMI upd sent  0
  LMI DLCI 1023  LMI type is CISCO  frame relay DTE
 Broadcast queue 0/64, broadcasts sent/dropped 1320/0, interface broadcasts 2
 Last input 00:00:00, output 00:00:00, output hang never
 Last clearing of "show interface" counters 22:01:52
 Input queue: 0/75/0 (size/max/drops); Total output drops: 0
Queueing strategy: weighted fair
 Output queue: 0/1000/64/0 (size/max total/threshold/drops)
  Conversations  0/1/256 (active/max active/max total)
    Reserved Conversations 0/0 (allocated/max allocated)
 5 minute input rate 0 bits/sec, 0 packets/sec
 5 minute output rate 0 bits/sec, 0 packets/sec
    12481 packets input, 720402 bytes, 0 no buffer
    Received 0 broadcasts, 0 runts, 0 giants, 0 throttles
```

```
      0 input errors, 0 CRC, 0 frame, 0 overrun, 0 ignored, 0 abort
       9579 packets output, 500221 bytes, 0 underruns
       0 output errors, 0 collisions, 0 interface resets
       0 output buffer failures, 0 output buffers swapped out
      0 carrier transitions
      DCD=up  DSR=up  DTR=up  RTS=up  CTS=up
Branch1#
```

LMI provides many of Frame Relay's useful features (such as keepalives) that can tell a router when one or more PVCs become unavailable. This example shows CISCO type LMI, which uses DLCI number 1023. If we had specified the CCITT or ANSI LMI standards, the router would use DLCI number 0 for LMI. Recipe 10.2 shows how to configure these different LMI options.

When you enable Frame Relay on an interface, the router automatically activates the Inverse ARP protocol, which is described in RFC 1293. The router uses Inverse ARP to make a dynamic mapping between a Frame Relay DLCI number and a Layer 3 address. This Layer 3 address could be for any supported protocol such as IP, AppleTalk, IPX, and so forth.

In this recipe, we built a static mapping between the DLCI number and the IP address, so we don't actually need Inverse ARP. Each subinterface always associates a particular DLCI number with a particular Layer 3 address. This means that we can safely disable Inverse ARP. You can do this for an individual protocol as follows:

```
Central(config)#interface Serial0
Central(config-if)#no frame-relay inverse-arp ip
```

Or you can disable Inverse ARP globally for all protocols:

```
Central(config)#interface Serial0
Central(config-if)#no frame-relay inverse-arp
```

In this case, if you want to reenable Inverse ARP just for a particular protocol you can do so:

```
Central(config)#interface Serial0
Central(config-if)#frame-relay inverse-arp ipx 100
```

This tells the router that it should use Inverse ARP to discover the IPX address of the device on the other end of the virtual circuit with DLCI number 100. If you don't need Inverse ARP, we generally recommend disabling it.

10.1.4 See Also

Recipe 10.2; Recipe 11.11

Top

Recipe 10.2 Adjusting LMI Options

10.2.1 Problem

You want to configure different LMI options on your Frame Relay circuit.

10.2.2 Solution

There are several different LMI options. The first specifies which version of LMI protocol you wish to use:

```
Branch1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Branch1(config)#interface Serial0
Branch1(config-if)#encapsulation frame-relay
Branch1(config-if)#frame-relay lmi-type ansi
Branch1(config-if)#end
Branch1#
```

By default, LMI sends keepalive packets through every PVC every 10 seconds to verify that the path is still available. You can adjust this value with the *keepalive* command:

```
Branch1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Branch1(config)#interface Serial0
Branch1(config-if)#encapsulation frame-relay
Branch1(config-if)#keepalive 5
Branch1(config-if)#end
Branch1#
```

LMI is not supported on all networks. If this is the case in your network, you must configure the router to announce its own DLCI number with the *local-dlci* command:

```
Branch1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Branch1(config)#interface Serial0
Branch1(config-if)#encapsulation frame-relay
Branch1(config-if)#frame-relay local-dlci 50
Branch1(config-if)#end
Branch1#
```

10.2.3 Discussion

The first example in this recipe sets an alternative LMI type. The default LMI type is called *cisco* on the router, although this is slightly confusing because it is not a Cisco proprietary standard, but rather was developed jointly by Cisco and other vendors. This default setting is usually the one you want, because it is the default setting in many Frame Relay switches.

The example shows how to set the ANSI standard LMI type:

```
Branch1(config-if)#frame-relay lmi-type ansi
```

Another LMI option, *q933a*, is also available. This option configures the router to use the Annex A of the ITU-T standard. This is sometimes called the CCITT LMI standard:

```
Branch1(config-if)#frame-relay lmi-type q933a
```

You must ensure that the LMI type you use matches what your carrier uses. It is not even necessary for all routers in the network to use the same LMI type, as long as each router matches the settings on its respective Frame Relay switch.

The second example sets the LMI keepalive time. Once again, this parameter depends on what is configured on the switch. In fact, as long as you use a keepalive setting that is less than the period on the switch, you should have no problems. In networks that do not use LMI, you should disable this polling:

```
Branch1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Branch1(config)#interface Serial0
Branch1(config-if)#encapsulation frame-relay
Branch1(config-if)#no keepalive
Branch1(config-if)#end
Branch1#
```

The last option defines a local DLCI:

```
Branch1(config-if)#frame-relay local-dlci 50
```

Here we have set the router's serial interface to use DLCI number 50. This command is required only on networks that do not use LMI, but there is no harm in configuring it for networks that do use LMI. In fact, some network engineers opt to configure this statement on all of their Frame Relay circuits as a mnemonic to remind them of the DLCI numbers that other devices use to reach this circuit. However, if you use the local DLCI number for this purpose, you should not disable keepalives—LMI requires them.

Top

Recipe 10.3 Setting Up Frame Relay with Map Statements

10.3.1 Problem

You want to configure Frame Relay services so that every PVC appears to share the same interface.

10.3.2 Solution

In its simplest form, the Frame Relay map configuration involves considerably less typing than the subinterface version of the same configuration:

```
Central#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Central(config)#interface Serial0
Central(config)#description Frame Relay to branches
Central(config-if)#ip address 192.168.1.1 255.255.255.0
Central(config-if)#encapsulation frame-relay
Central(config-if)#frame-relay map ip 192.168.1.10 101
Central(config-if)#frame-relay map ip 192.168.1.11 102
Central(config-if)#frame-relay map ip 192.168.1.12 103
Central(config-if)#end
Central#
```

10.3.3 Discussion

Instead of treating the Frame Relay WAN as a series of point-to-point logical connections as we did in Recipe 10.1, you can configure it to look similar to a LAN segment with a contiguous block of IP addresses. There are two ways to do this, either by using frame-relay map statements as in this recipe, or by using multipoint subinterfaces as in Recipe 10.4. In general, we prefer to use point-to-point subinterfaces for Frame Relay networks because it gives you more detailed controls over the routing protocol.

Furthermore, when you use point-to-point subinterfaces, the router can generate a trap when a DLCI becomes inactive, which makes network management much easier. With multipoint subinterfaces, the router will generate a trap only when all of the associated DLCIs become unavailable, but not for individual failures. And when you use frame-relay map statements, the router will not provide any notification of DLCI failures. However, while the Frame Relay map method lacks some of the features of subinterfaces, it is still perfectly acceptable in less complicated networks.

The *frame-relay map* command has several useful options. In the example, we used the simplest version of the command:

```
Central(config-if)#frame-relay map ip 192.168.1.10 101
```

This associates the IP address `192.168.1.10` with DLCI number 101. If you have other protocols, such as IPX or AppleTalk, you must configure them separately:

```
Central(config-if)#frame-relay map ipx 10AF.0.0.1 101
Central(config-if)#frame-relay map appletalk 1.15 101
```

The configuration of the router on the other end of this PVC can use either a similar map statement, or a subinterface, as in Recipe 10.1. If you use a map statement on the remote router, it should be configured with the IP address and DLCI number for the router on this end:

```
Branch1(config-if)#frame-relay map ip 192.168.1.1 50
```

When you configure a Frame Relay map like this, you create a static mapping between a DLCI number and a Layer 3 protocol address, IP in this case. In Recipe 10.1 we mentioned that when you create a static mapping, you can disable Inverse ARP. If you don't need the router to make dynamic associations between Frame Relay DLCI numbers and Layer 3 addresses, we recommend disabling Inverse ARP:

```
Central(config)#interface Serial0
Central(config-if)#no frame-relay inverse-arp
```

When you set up a TCP/IP network using map statements, you have to bear in mind that the Frame Relay network does not handle broadcasts to the remote sites like a LAN segment. In fact, it is the classic example of a Non-Broadcast Multiple Access (NBMA) network. This can make OSPF configuration somewhat more complex, and it can cause serious problems for routing protocols that don't handle NBMA media well.

The default configuration for most routing protocols assumes that you can reach all of the adjacent routers through a particular interface with either a broadcast or a multicast advertisement packet. With some protocols, you can configure the neighbors statically and instruct them to use unicast packets to exchange routing information instead. However, if you leave this in the default configuration, the routing protocols will not work at all.

Cisco routers also allow you to cheat a little bit and treat the Frame Relay network as if it were a broadcast medium by simply adding the *broadcast* keyword to the map statement:

```
Central(config-if)#frame-relay map ip 192.168.1.10 101 broadcast
```

In this case, if you were to send a packet to the broadcast address, `192.168.1.255`, the router would make copies of the broadcast packet and send it out to all of the remote sites configured on this interface. Please refer to Chapter 6, Chapter 7, and Chapter 8 for more information on routing protocols.

The *ietf* keyword is another useful option in some situations. This tells the router to use RFC 1490 encapsulation instead of the default Cisco proprietary encapsulation:

```
Central(config-if)#frame-relay map ip 192.168.1.10 101 ietf
```

The default encapsulation does not work well with equipment from some vendors. So, if you have problems connecting to non-Cisco equipment, this option might help. You can also enable RFC 1490 encapsulation globally on the whole interface as follows:

```
Central(config)#interface Serial0
Central(config-if)#encapsulation frame-relay ietf
Central(config-if)#end
```

In this case, it is no longer necessary to specify the *ietf* keyword on each map statement. However, if you want to revert to the default Cisco encapsulation on a particular PVC, you can use the *cisco* keyword:

```
Central(config-if)#frame-relay map ip 192.168.1.10 101 cisco
```

We discuss some more options for the *frame-relay map* command in Recipe 10.8.

10.3.4 See Also

Recipe 10.1; Recipe 10.8; Recipe 10.9; Chapter 6; Chapter 7;

Top

Recipe 10.4 Using Multipoint Subinterfaces

10.4.1 Problem

You want to configure Frame Relay so that many PVCs share the same subinterface.

10.4.2 Solution

You can connect several virtual circuits to a single subinterface as follows:

```
Central#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Central(config)#interface Serial0.1 multipoint
Central(config-subif)#description Frame Relay to branches
Central(config-subif)#ip address 192.168.1.1 255.255.255.0
Central(config-subif)#frame-relay interface-dlci 101
Central(config-subif)#frame-relay interface-dlci 102
Central(config-subif)#frame-relay interface-dlci 103
Central(config-subif)#frame-relay interface-dlci 104
Central(config-subif)#end
Central#
```

10.4.3 Discussion

Recipe 10.1 showed how to create a separate subinterface for each Frame Relay DLCI. Recipe 10.3 showed how to configure all of the DLCIs to share the same interface and the same address range. This recipe shows a method that is somewhere in between these two extremes, with several virtual circuits sharing a common subinterface. You can even combine these multipoint subinterfaces with point-to-point subinterfaces on the same physical interface if you wish to create a hybrid of the two styles.

When you use a multipoint configuration, the subinterface will appear to be active unless all of the DLCIs associated with it become unavailable. If this happens, the subinterface will go into a down state and the router will send a trap. This is different from the behavior when using Frame Relay maps on the physical interface, as discussed in Recipe 10.3. In that case, the interface will change state only if the signaling on the physical interface fails. For this reason, we recommend using multipoint configuration instead of Frame Relay maps.

As an interesting aside, you can configure Frame Relay maps on a multipoint subinterface. This could be useful if you need to mix point-to-point and multipoint interface types on a network that doesn't

support Inverse ARP.

As we mentioned in Recipe 10.1, it is not a simple matter to change the subinterface type between point-to-point and multipoint. If you have already defined a subinterface as one type and you want to change it to the other, you must delete the subinterface and reboot the router. Then you can add the subinterface configuration back to the router with the correct type.

The biggest difference between using multipoint subinterfaces and any of the previous examples is that here we do not configure a static mapping between a particular DLCI number and a remote IP address. So, to make this association, the router must use Inverse ARP. If you disable Inverse ARP here, as we did in the previous recipes, the network will not work.

You can see that Inverse ARP is correctly mapping IP addresses to DLCI numbers with the *show frame-relay map* command:

```
Central#show frame-relay map
Serial0.1 (up): ip 192.168.55.6 dlci 100(0x64,0x1840), dynamic,
              broadcast,, status defined, active
Central#
```

If we reconfigure this same DLCI to use a point-to-point subinterface instead (which requires rebooting the router, as we mentioned earlier), you see that the output of the show frame-relay map command changes significantly to tell you about the static mapping:

```
Central#show frame-relay map
Serial0.1 (up): point-to-point dlci, dlci 100(0x64,0x1840), broadcast
          status defined, active
Central#
```

10.4.4 See Also

Recipe 10.1; Recipe 10.3; Recipe 10.5

Top

Recipe 10.5 Configuring Frame Relay SVCs

10.5.1 Problem

You want to configure the router to support Frame Relay SVCs.

10.5.2 Solution

Frame Relay SVCs are not extremely common, but some carrier networks support them. The advantage to using SVCs is that the router can add and remove inactive virtual circuits dynamically in a lightly used network. Because of the extra complexity and the management problems associated with dynamic network topologies, most network engineers will only use this feature if it offers significant cost advantages.

You can configure SVCs to use subinterfaces as in Recipe 10.1:

```
Central#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Central(config)#interface Serial0
Central(config-if)#encapsulation frame-relay
Central(config-if)#frame-relay lmi-type q933a
Central(config-if)#frame-relay svc
Central(config-if)#exit
Central(config)#interface Serial0.10 point-to-point
Central(config-subif)#ip address 192.168.1.129 255.255.255.252
Central(config-subif)#frame-relay interface-dlci 100
Central(config-subif)#map-group SVCMAP
Central(config-fr-dlci)#class SVCclass
Central(config-fr-dlci)#exit
Central(config-subif)# exit
Central(config)#map-list SVCMAP source-addr X121 1234 dest-addr X121 4321
Central(config-map-list)#ip 192.168.55.6 class SVCclass ietf
Central(config-map-list)#exit
Central(config)#map-class frame-relay SVCclass
Central(config-map-class)#frame-relay traffic-rate 56000 128000
Central(config-map-class)#end
Central#
```

You can also configure Frame Relay SVCs using map statements, similar to Recipe 10.3:

```
Central#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
```

```
Central(config)#interface Serial
Central(config-if)#ip address 192.168.55.1 255.255.255.0
Central(config-if)#encapsulation frame-relay
Central(config-if)#frame-relay lmi-type q933a
Central(config-if)#frame-relay svc
Central(config-if)#map-group SVCMAP
Central(config-if)#frame-relay interface-dlci 50
Central(config-fr-dlci)#class SVCclass
Central(config-fr-dlci)#exit
Central(config-if)#exit
Central(config)#map-list SVCMAP source-addr X121 1234 dest-addr X121 4321
Central(config-map-list)#ip 192.168.55.6 class SVCclass ietf
Central(config-map-list)#exit
Central(config)#map-class frame-relay SVCclass
Central(config-map-class)#frame-relay traffic-rate 56000 128000
Central(config-map-class)#end
Central#
```

10.5.3 Discussion

You can enable Frame Relay SVCs on an interface simply by including the *frame-relay svc* command. This is required whether you use maps or subinterfaces:

```
Central(config-if)#frame-relay svc
```

However, this doesn't tell the network how to actually build the virtual circuits. To do that, you need to define a map list and a map class:

```
Central(config)#map-list SVCMAP source-addr X121 1234 dest-addr X121 4321
Central(config-map-list)#ip 192.168.55.6 class SVCclass ietf
Central(config-map-list)#exit
Central(config)#map-class frame-relay SVCclass
Central(config-map-class)#frame-relay traffic-rate 56000 128000
Central(config-map-class)#end
```

The map list associates an IP address with either X.121 or E.164 source and destination addresses. We have used X.121 addresses in the example, but if your carrier's network uses E.164 addressing instead, you can simply replace the keyword *X121* with *E164* and specify the appropriate E.164 addresses:

```
Central(config)#map-list SVCMAP source-addr E164 1234 dest-addr E164 4321
```

The map-class command tells the router about the actual SVC parameters such as CIR and EIR. In this example, we want the network to create SVCs with CIR of 56,000 and total burst rate (CIR+EIR) of 128,000 bits per second.

By default, the router will keep an idle SVC for 120 seconds before tearing it down. You can change this period using the *frame-relay idle-timer* command. There are three ways to specify an idle time. You can have the router tear down an idle PVC if there is no traffic in either direction for a specified time period like this:

```
Central(config)#map-class frame-relay SVCclass
Central(config-map-class)#frame-relay idle-timer 60
```

Or you can specify the inbound and outbound directions separately:

```
Central(config)#map-class frame-relay SVCclass
Central(config-map-class)#frame-relay idle-timer in 20
Central(config-map-class)#frame-relay idle-timer out 30
```

In each case, the argument is the time period specified in seconds.

You can view the SVC map information on a router with the *show frame-relay svc* command:

```
Central#show frame-relay svc maplist SVCMAP
Map List : SVCMAP
Address : Source X121 1234 <----> Destination X121 4321

Protocol : ip 192.168.55.6                          Encapsulation : IETF

FMIF (Frame Mode Information Field Size), bytes
Configured : In = 1500, Out = 1500

CIR (Committed Information Rate), bits/sec
Configured : In = 56000,        Out = 56000,

Minimum Acceptable CIR, bits/sec
Configured : In = 56000,        Out = 56000,

Bc (Committed Burst Size), bits
Configured : In = 56000,        Out = 56000,

Be (Excess Burst Size), bits
Configured : In = 56000,        Out = 56000,

Central#
```

Remember that whether you use maps or subinterfaces, you can combine SVCs and PVCs on the same physical interface.

10.5.4 See Also

Recipe 10.1; Recipe 10.3

Top

Recipe 10.6 Simulating a Frame Relay Cloud

10.6.1 Problem

You want to use a router to simulate a Frame Relay cloud in the lab.

10.6.2 Solution

A Cisco router can function as a Frame Relay switch. This is mostly useful when you are trying to simulate a Frame Relay cloud in a lab to test your router configurations:

```
Cloud#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Cloud(config)#frame-relay switching
Cloud(config)#interface Serial0
Cloud(config-if)#description Frame-relay connection to Central - DLCI 50
Cloud(config-if)#encapsulation frame-relay
Cloud(config-if)#clock rate 125000
Cloud(config-if)#frame-relay lmi-type cisco
Cloud(config-if)#frame-relay intf-type dce
Cloud(config-if)#frame-relay route 101 interface Serial1 50
Cloud(config-if)#frame-relay route 102 interface Serial2 50
Cloud(config-if)#exit
Cloud(config)#interface Serial1
Cloud(config-if)#description Frame-relay connection to Branch1 - DLCI 101
Cloud(config-if)#encapsulation frame-relay
Cloud(config-if)#clock rate 125000
Cloud(config-if)#frame-relay lmi-type cisco
Cloud(config-if)#frame-relay intf-type dce
Cloud(config-if)#frame-relay route 50 interface Serial0 101
Cloud(config-if)#exit
Cloud(config)#interface Serial2
Cloud(config-if)#description Frame-relay connection to Branch2 - DLCI 102
Cloud(config-if)#encapsulation frame-relay
Cloud(config-if)#clock rate 125000
Cloud(config-if)#frame-relay lmi-type cisco
Cloud(config-if)#frame-relay intf-type dce
Cloud(config-if)#frame-relay route 50 interface Serial0 102
Cloud(config-if)#end
Cloud#
```

10.6.3 Discussion

This type of configuration can be extremely useful when you need to test basic Frame Relay functionality in a lab, but don't have a real Frame Relay switch available. However, it's extremely important to remember that a router is not a Frame Relay switch, and it doesn't emulate all of the functionality of the switch. In particular, no matter how congested you conspire to make the router, it will never generate FECN or BECN notifications. So, if you are using this type of configuration to test adaptive traffic shaping (or any other feature that relies on BECN notifications), it will not give you a reliable simulation of a real cloud.

To use the router as a Frame Relay switch, you must first enable the *frame-relay switching* option. Then you must configure each interface as DCE with the *frame-relay intf-type* command and supply a clock signal with the *clock rate* command. Cisco routers will not allow you to configure this command unless you use a DCE cable on the interface. Finally, you need to map the PVCs. In this case, we have configured a central hub router and two branch routers, as in Recipe 10.1. The central router can see both of the branch routers, one with DLCI 101, and the other with DLCI 102. Both of the branch routers see the central router with DLCI 50. The two branch routers cannot see one another directly.

In this example, all three of the Frame Relay connections are to DTE devices such as routers, so all of the interfaces are configured for DCE signaling. However, you can also configure connections to other switching devices. This might be useful if you were interested in constructing your own private Frame Relay cloud. In this case, you would still need to designate one of the devices to be the physical DCE and supply the clock. Then you would configure the interface type on both devices as Network to Network Interfaces (NNI) with the *nni* keyword:

```
Cloud#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Cloud(config)#interface Serial2
Cloud(config-if)#description Frame-relay connection to next switch
Cloud(config-if)#encapsulation frame-relay
Cloud(config-if)#clock rate 125000
Cloud(config-if)#frame-relay lmi-type cisco
Cloud(config-if)#frame-relay intf-type nni
Cloud(config-if)#end
Cloud(config)#
```

You would also use *frame-relay route* statements to configure one or more PVCs to be served by this neighboring switch. The PVC routing commands in this case are identical to those for DCE interfaces.

You can look at the routing of the virtual circuits on a router that is configured for Frame Relay switching with the *show frame-relay route* command:

```
Cloud#show frame-relay route
Input Intf      Input Dlci      Output Intf     Output Dlci     Status
Serial0         101             Serial1         50              active
Serial0         102             Serial2         50              inactive
Serial0         103             Serial3         50              inactive
Serial1         50              Serial0         101             active
Serial1         102             Serial2         101             inactive
```

```
Serial1          103              Serial3          101              inactive
Serial2          50               Serial0          102              inactive
Serial2          101              Serial1          102              inactive
Serial2          103              Serial3          102              inactive
Serial3          50               Serial0          103              inactive
Serial3          101              Serial1          103              inactive
Serial3          102              Serial2          103              inactive
Cloud#
```

This output shows, for example, that traffic received on DLCI number 101 through interface `Serial0` is forwarded to DLCI number 50 on `Serial1`. And, a few lines lower you can see the reverse path as well. The status for both of these lines is `active`, so this virtual circuit is working properly.

10.6.4 See Also

Recipe 10.1

Top

Recipe 10.7 Compressing Frame Relay Data on a Subinterface

10.7.1 Problem

You want to configure your router to do Frame Relay compression on a subinterface.

10.7.2 Solution

Cisco offers several different types of compression with Frame Relay. You can opt to compress only the TCP headers as follows:

```
Central#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Central(config)#interface Serial0
Central(config-if)#encapsulation frame-relay
Central(config-if)#frame-relay ip tcp header-compression passive
Central(config-if)#end
Central#
```

This command also works at the subinterface level:

```
Central#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Central(config)#interface Serial0.1 point-to-point
Central(config-subif)#frame-relay ip tcp header-compression passive
Central(config-subif)#end
Central#
```

There are also two different payload compression options. The first uses the FRF.9 Frame Relay compression standard:

```
Central#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Central(config)#interface Serial0.1 point-to-point
Central(config-if)#frame-relay payload-compression frf9 stac
Central(config-if)#end
Central#
```

The second uses Cisco's proprietary packet-by-packet compression:

```
Central#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Central(config)#interface Serial0.1 point-to-point
```

```
Central(config-if)#frame-relay payload-compression packet-by-packet
Central(config-if)#end
Central#
```

10.7.3 Discussion

The nice thing about the first example in this recipe is that, with the *passive* keyword, the router will send packets with compressed TCP headers only if it receives packets with compressed headers. So if you have a variety of remote sites, some of which have routers that don't support header compression, this can be a useful configuration option. You need to configure the device at least one end without the *passive* keyword:

```
Branch1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Branch1(config)#interface Serial0
Branch1(config-if)#encapsulation frame-relay
Branch1(config-if)#frame-relay ip tcp header-compression
Branch1(config-if)#end
Branch1#
```

Note that Cisco recommends shutting down the interface before changing this feature. It is not dangerous, but with some routers you need to reset the interface to ensure that it picks up the new configuration. The cleanest way to do this is to shut it down before making the change, then bring it back up when you are done.

For the payload compression examples, it is critical to configure the same compression algorithm on both ends. This is a subinterface level command, so you can configure each PVC to use compression or not according to what the device on the other end supports.

By default, the router will do the compression in a Compression Service Adapter (CSA) if one exists. If the router doesn't have a CSA, then it will use a Versatile Interface Processor (VIP-2) card instead. And, if it doesn't have either of these hardware options, then it will do the compression in software using the router's CPU. Some external Frame Relay Access Devices (FRAD) also include FRF.9 compression, but it is unlikely that you will find a FRAD that supports Cisco's packet-by-packet compression.

The FRF.9 compression command can also take several different options that allow you to force different hardware options. For example, you can force the router to use a particular CSA:

```
Central(config-if)#frame-relay payload-compression frf9 stac csa  1
```

Or, if you want to force the router to do the compression in its CPU, you can use the software keyword:

```
Central(config-if)#frame-relay payload-compression frf9 stac software
```

The *stac* keyword in all of these FRF.9 examples specifies the standard Stacker algorithm. In fact, this is the only option available for FRF.9 compression.

In general, we recommend using FRF.9 rather than packet-by-packet compression because it is an open standard, while packet-by-packet will only work with Cisco equipment. There is no noticeable performance difference between the two compression types. Cisco introduced its own packet-by-packet compression method before the FRF.9 standard was available, and continues to support it primarily for backward compatibility.

You can see statistics on the header compression with the following command:

```
Router#show frame-relay ip tcp header-compression
  DLCI 100        Link/Destination info:  point-to-point dlci
  Interface Serial1:
    Rcvd:    220 total, 219 compressed, 0 errors
             0 dropped, 0 buffer copies, 0 buffer failures
    Sent:    482 total, 481 compressed,
             17001 bytes saved, 229749 bytes sent
             1.7 efficiency improvement factor
  Connect: 16 rx slots, 16 tx slots, 1 long searches, 1 misses
             99% hit ratio, five minute miss rate 0 misses/sec, 0 max

Router#
```

10.7.4 See Also

Recipe 10.9; Recipe 10.8

Top

Recipe 10.8 Compressing Frame Relay Data with Maps

10.8.1 Problem

You want to configure your router to do Frame Relay compression with map statements.

10.8.2 Solution

The same Frame Relay compression options that we discussed for subinterfaces are also available with map statements. You can turn on FRF.9 compression by simply including a few additional keywords in the *frame-relay map* statement as follows:

```
Central#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Central(config)#interface Serial0
Central(config-if)#description Frame Relay to branches
Central(config-if)#ip address 192.168.1.1 255.255.255.0
Central(config-if)#encapsulation frame-relay
Central(config-if)#frame-relay map ip 192.168.1.10 101  payload-compression frf9 stac
Central(config-if)#end
Central#
```

Or you can opt to use Cisco's proprietary packet-by-packet compression instead:

```
Central#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Central(config)#interface Serial0
Central(config-if)#description Frame Relay to branches
Central(config-if)#ip address 192.168.1.1 255.255.255.0
Central(config-if)#encapsulation frame-relay
Central(config-if)#frame-relay map ip 192.168.1.10 101  payload-compression packet-by-
packet
Central(config-if)#end
Central#
```

The map configuration also supports TCP header compression:

```
Central#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Central(config)#interface Serial0
Central(config-if)#description Frame Relay to branches
Central(config-if)#ip address 192.168.1.1 255.255.255.0
Central(config-if)#encapsulation frame-relay
Central(config-if)#frame-relay map ip 192.168.1.10 101  compress
Central(config-if)#end
Central#
```

10.8.3 Discussion

As we discussed in Recipe 10.7, Cisco routers are able to compress the data payload of packets before sending them through Frame Relay circuits. This recipe simply shows how to do the same thing using map statements instead of subinterfaces. Note that the header compression example shown in Recipe 10.7 applies to the physical interface, so the configuration for header compression is identical whether you are using maps or subinterfaces.

You can combine the payload compression option with the other options we discussed in Recipe 10.3 by specifying all of the required options on the same command line:

```
Central(config)#interface Serial0
Central(config-if)#encapsulation frame-relay
Central(config-if)#frame-relay map ip 192.168.1.10 101 ietf broadcast payload-
compression frf9 stac
Central(config-if)#end
```

Note that you can specify these keywords in any order, as long as the compression options are last. However, you cannot combine the different compression options with one another.

10.8.4 See Also

Recipe 10.3 ; Recipe 10.7

Top

Recipe 10.9 Viewing Frame Relay Status Information

10.9.1 Problem

You want to check the status of a Frame Relay circuit or VC.

10.9.2 Solution

There are several useful show commands for looking at Frame Relay circuits and virtual circuits. It is usually best to start at the physical layer and work upward through the protocol layers. You can look at the physical interface with the *show interfaces* command:

```
Central#show interfaces serial
```

The *show frame-relay pvc* command allows you to see information about each of your Frame Relay PVCs:

```
Central#show frame-relay pvc
```

And sometimes it is also useful to look at the LMI status:

```
Central#show frame-relay lmi
```

10.9.3 Discussion

The *show interfaces* command has a lot of useful information. When the interface is configured for Frame Relay, this command shows the LMI configuration, whether the interface is configured for SVCs as well as PVCs, and whether the interface is set up to be DCE or DTE. But the most important thing to look at is always the first line, which shows the physical and the protocol statuses:

```
Central#show interfaces serial
Serial0 is up, line protocol is up
  Hardware is HD64570
  Description: Frame Relay connection
  MTU 1500 bytes, BW 1544 Kbit, DLY 20000 usec,
     reliability 255/255, txload 3/255, rxload 3/255
  Encapsulation FRAME-RELAY, loopback not set, keepalive set (10 sec)
  LMI enq sent  263, LMI stat recvd 263, LMI upd recvd 0, DTE LMI up
  LMI enq recvd 0, LMI stat sent  0, LMI upd sent  0
  LMI DLCI 0  LMI type is CCITT  frame relay DTE
  FR SVC enabled, LAPF state down
```

```
Broadcast queue 0/64, broadcasts sent/dropped 44/0, interface broadcasts 0
Last input 00:00:03, output 00:00:03, output hang never
Last clearing of "show interface" counters never
Input queue: 0/75/0 (size/max/drops); Total output drops: 0
Queueing strategy: weighted fair
Output queue: 0/1000/64/0 (size/max total/threshold/drops)
    Conversations  0/2/256 (active/max active/max total)
    Reserved Conversations 0/0 (allocated/max allocated)
5 minute input rate 24000 bits/sec, 0 packets/sec
5 minute output rate 23000 bits/sec, 0 packets/sec
    2838 packets input, 1604468 bytes, 0 no buffer
    Received 0 broadcasts, 0 runts, 0 giants, 0 throttles
   0 input errors, 0 CRC, 0 frame, 0 overrun, 0 ignored, 0 abort
    2951 packets output, 1623730 bytes, 0 underruns
    0 output errors, 0 collisions, 20 interface resets
    0 output buffer failures, 0 output buffers swapped out
   2 carrier transitions
    DCD=up  DSR=up  DTR=up  RTS=up  CTS=up
Central#
```

If the interface is up you should be able to see useful PVC information:

```
Central#show frame-relay pvc

PVC Statistics for interface Serial1 (Frame Relay DTE)

DLCI = 100, DLCI USAGE = LOCAL, PVC STATUS = ACTIVE, INTERFACE = Serial0.1

  input pkts 1271          output pkts 1312         in bytes 843519
  out bytes 856138         dropped pkts 0           in FECN pkts 0
  in BECN pkts 0           out FECN pkts 0          out BECN pkts 0
  in DE pkts 0             out DE pkts 0
  out bcast pkts 40         out bcast bytes 11320
  pvc create time 01:08:11, last time pvc status changed 00:39:42
Central#
```

This output tells you, for example, that the PVC with DLCI 100 is active and configured on subinterface `Serial0.1`. None of the packets received on this subinterface have had their FECN, BECN, or DE bits set. This is the most useful place to check for congestion in the Frame Relay cloud. Note that the router is unlikely to ever set the FECN or BECN bits when sending packets, so the inbound counters are the most useful here.

The last line of this output for each PVC is particularly useful if you have a problem with flapping PVCs in the carrier cloud. In this case, you can see that the PVC has been active for just over an hour, but it had a status change 39 minutes ago. This doesn't tell you what caused the status change, though. In a stable network, you should not expect to see frequent PVC status changes. So this gives you a useful indication of problems either in the carrier cloud, or with your remote router.

Note that the *show frame-relay pvc* command will list all of the PVCs on a router, including any that

are configured on the router but not in use. It will also list any PVCs that are configured on the switch, but not on the router. If you want to focus on a particular PVC, you can specify the one you want by its DLCI number:

```
Central#show frame-relay pvc 100
```

If you suspect an LMI problem, it is useful to look at the output of the *show frame-relay lmi* command:

```
Central#show frame-relay lmi

LMI Statistics for interface Serial1 (Frame Relay DTE) LMI TYPE = CCITT
  Invalid Unnumbered info 0                 Invalid Prot Disc 0
  Invalid dummy Call Ref 0                  Invalid Msg Type 0
  Invalid Status Message 0                  Invalid Lock Shift 0
  Invalid Information ID 0                  Invalid Report IE Len 0
  Invalid Report Request 0                  Invalid Keep IE Len 0
  Num Status Enq. Sent 299                  Num Status msgs Rcvd 299
  Num Update Status Rcvd 0                  Num Status Timeouts 0
Central#
```

The first line of this output shows that the LMI type in this case is CCITT, which is configured with the *frame-relay lmi-type q933a* command. The other options are *cisco* and *ansi*, which both use the same type field in the output of this command as in the configuration command.

Because LMI is the Frame Relay management protocol between the router and the switch, the usual symptom of an LMI problem is that the physical interface is up, but the protocol is down and none of the PVCs will come up. If you repeatedly check the *show frame-relay lmi* command, you will see the `Num Status Timeouts` field incrementing. Because it can take several seconds for an interface to come up, it is sometimes hard to tell immediately if you have the right LMI type field. This field gives you a relatively quick indication of when you have the right configuration.

Top

Chapter 11. Queueing and Congestion

Top

Introduction

Quality of Service (QoS) has been a part of the IP protocol since RFC 791 was released in 1981. However, it has not been extensively used until recently. The main reason for using QoS in an IP network is to protect sensitive traffic in congested links. In many cases, the best solution to the problem of congested links is simply to upgrade them. All you can do with a QoS system is affect which packets are forwarded and which ones are delayed or dropped when congestion is encountered. This is effective only when the congestion is intermittent. If a link is just consistently over- utilized, then QoS will at best offer a temporary stopgap measure until the link is upgraded or the network is redesigned.

There are several different traffic flow characteristics that you can try to control with a QoS system. Some applications require a certain minimum throughput to operate, while others require a minimum latency. Jitter, which is the difference in latency between consecutive packets, has to be carefully constrained for many real-time applications such as voice and video. Some applications do not tolerate dropped packets well. Others contain time-sensitive information that is better dropped than delayed.

There are essentially three steps to any traffic prioritization scheme. First, you have to know what your traffic patterns look like. This means you need to understand what traffic is mission critical, what can wait, and what traffic flows are sensitive to jitter, latency, or have minimum throughput requirements. Once you know this, the second step is to provide a way to identify the different types of traffic. Usually, in IP QoS you will use this information to tag the Type of Service (TOS) byte in the IP header. This byte contains a 6-bit field called the Differentiated Services Control Point (DSCP) in newer literature, and is separated into a 3-bit IP Precedence field and a TOS field (either 3 or 4 bits) in older literature. These fields are used for the same purpose, although there are differences in their precise meanings. We discuss these fields in more detail in Appendix B.

The third step is to configure the network devices to use this information to affect how the traffic is actually forwarded through the network. This is the step where you actually have the most freedom, because you can decide precisely what you want to do with different traffic types. However there are two main philosophies here: TOS-based routing and DSCP per-hop behavior.

TOS-based routing basically means that the router selects different paths based on the contents of the TOS field in the IP header. However, the precise TOS behavior is left up to the network engineer, so the TOS values could affect other things such as queueing behavior. DSCP, on the other hand, generally looks at the same set of bits and uses them to decide how to handle the queueing when the links are congested. TOS-based routing is the older technique, and DSCP is newer.

You can easily implement TOS-based routing to select different network paths using Cisco's Policy Based Routing (PBR). For example, some engineers use this technique of Frame Relay networks to

funnel high priority traffic into a different PVC than lower priority traffic. And many standard IP protocols such as FTP and Telnet have well-defined default TOS settings.

Most engineers prefer the DSCP approach because it is easier to implement and troubleshoot. If high priority application packets take a different path than low priority PING packets, as is possible in the TOS approach, it can be extremely confusing to manage the network. DSCP is also usually easier to implement and less demanding of the router's CPU and memory resources, as well as being more consistent with the capabilities of modern routing protocols.

Note that any time you stop a packet to examine it in more detail, you introduce latency and potentially increase the CPU load on the router. The more fields you examine or change, the greater the impact. For this reason, we want to stress that the best network designs handle traffic prioritization by marking the packets as early as possible. Then other routers in the network only need to look at the DSCP field to handle the packet correctly. In general, you want to keep this marking function at the edges of the network where the traffic load is lowest, rather than in the core where the routers are too busy forwarding packets to examine and classify packets.

We discuss the IP Precedence, TOS, and DSCP classification schemes in more detail in Appendix B.

Queueing Algorithms

The simplest type of queue transmits packets in the same order that it receives them. This is called a First In First Out (FIFO) queue. And, although it sounds naively like it treats all traffic streams equally, it actually tends to favor resource-hungry, ill-behaved applications.

The problem is that if a single application sends a burst that fills a FIFO queue, the router will wind up transmitting most of the queued packets, but will have to drop incoming packets from other applications. If these other applications adapt to the decrease in available bandwidth by sending at a slower rate, the ill-behaved application will greedily take up the slack and could gradually choke off all of the other applications.

Because FIFO queueing allows some data flows to take more than their share of the available bandwidth, it is called *unfair*. Fair Queueing (FQ) and Weighted Fair Queueing (WFQ) are two of the simpler algorithms that have been developed to deal with this problem. Both of these algorithms sort incoming packets into a series of *flows*.

We discuss Cisco's implementations of different queueing algorithms in Appendix B.

When talking about queueing, it is easy to get wrapped up in relative priorities of data streams. However, it is just as important to think about how your packets should be dropped when there is congestion. Cisco routers allow you to even implement a congestion avoidance system called Random Early Detection (RED), which also has a weighted variant, Weighted Random Early Detection (WRED). These algorithms allow the router to start dropping packets before there is a serious

congestion problem. This forces well-behaved TCP applications to back off and send their data more slowly, thereby avoiding congestion problems before they start. RED and WRED are also discussed in Appendix B.

Fast Switching

One of the most important performance limitations on a router depends on how the packets are processed internally. The worst case is where the router's CPU has to examine every packet to decide how to forward it. Packets that are handled in the CPU like this are said to use *process switching*. It is never possible to completely eliminate process switching in a router, because the router has to react to some types of packets, particularly those containing network control information. And, as we will discuss in a moment, process switching is often used to bootstrap other more efficient methods.

For many years, Cisco has included more efficient methods for packet processing in routers. These often involve offloading the routing decisions to special logic circuits, frequently associated with interface hardware. The actual details of how these circuits work is often not of much interest to the network engineer. The most important thing is to ensure that as many packets as possible use these more efficient methods.

Fast switching is one of Cisco's earlier mechanisms for offloading routing from the CPU. In fast switching, the router uses process switching to forward the first packet to a particular destination. The CPU looks up the appropriate forwarding information in the routing table and then sends the packet accordingly. Then, when the router sees subsequent packets for the same destination, it is able to use the same forwarding information. Fast switching records this forwarding information in an internal cache, and uses it to bypass the laborious route lookup process for all but the first packet in a flow. It works best when there is a relatively long stream of packets to the same destination. And, of course, it is necessary to periodically verify that the same forwarding information is still valid. So fast switching requires the router to process switch some packets just to check that the cached path is still the best path.

To allow for reliable load balancing, the fast switching cache includes only /32 addresses. This means that there is no network or subnet level summarization in this cache. Whenever the fast switching algorithm receives a packet for a destination that is not in its cache, or that it can't handle because of a special filtering feature that isn't supported by fast switching, it must *punt*. This means that the router passes the packet to a more general routing algorithm, usually process switching.

Fast switching works only with active traffic flows. A new flow will have a destination that is not in the fast switching cache. Similarly, low-bandwidth applications that only send one packet at a time, with relatively long periods between packets, will not benefit from fast switching. In both of these cases, the router must punt, and process switch the packet. Another more serious example happens in busy Internet routers. These devices have to deal with so many flows that they are unable to cache them all.

Largely because of this last problem, Cisco developed a more sophisticated system called Cisco Express Forwarding (CEF) that improves on several of the shortcomings of fast switching. The main improvement is that instead of just caching active destinations, CEF caches the entire routing table. This increases the amount of memory required, but the routing information is stored in an efficient, hashed structure.

The router keeps the cached table synchronized with the main routing table that is acquired through a dynamic routing protocol such as OSPF or BGP. This means that CEF needs to punt a packet only when it requires features that don't work with CEF. For example, some policy-based routing rules do not work with CEF. So, when you use these, CEF must still punt and process switch these packets.

In addition to caching the entire routing table, CEF also maintains a table of information about all available next-hop devices. This allows the router to build the appropriate Layer 2 framing information for packets that need to be forwarded, without having to consult the system ARP table.

Because CEF rarely needs to punt a packet, even if it is the first packet of a new flow, it is able to operate much more efficiently than fast switching. And because it caches the entire routing table, it is even able to do packet-by-packet round-robin load sharing between equal cost paths. CEF shows its greatest advantage over fast switching in situations where there are many flows, each relatively short in duration. Another key advantage is that CEF has native support for QoS, while fast switching does not.

A Distributed CEF is available on routers that support Versatile Interface Processor (VIP) cards, such as the 7500 series. This allows each VIP card to run CEF individually to further improve scalability.

Top

Recipe 11.1 Fast Switching and CEF

11.1.1 Problem

You want to use the most efficient mechanism in the router to switch the packets.

11.1.2 Solution

As we discuss in Appendix B, one of the most important things you can do to improve router performance, and consequently network performance, is to ensure that you are using the best packet switching algorithm. All Cisco routers support fast switching, and it is enabled by default. However, some types of configurations require that it be disabled. The following example shows how to turn fast switching back on if it has been disabled:

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#interface FastEthernet0/0
Router(config-if)#ip route-cache
Router(config-if)#end
Router#
```

If you are using policies, including policies for class-based QoS, you also need to configure fast switching to handle them, using the *ip route-cache policy* command:

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#interface FastEthernet0/0
Router(config-if)#ip route-cache policy
Router(config-if)#end
Router#
```

CEF, on the other hand, is not enabled by default. Unlike fast switching, which is enabled separately for each interface, you have to enable CEF globally for the entire router, as well as on each interface:

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#ip cef
Router(config)#interface FastEthernet0/0
Router(config-if)#ip route-cache cef
Router(config-if)#end
Router#
```

11.1.3 Discussion

The *ip route-cache* command used to enable fast switching has a couple of useful options. The second example demonstrates one of these, the *policy* keyword, which allows fast switching of policy-based routing:

```
Router(config-if)#ip route-cache policy
```

Another useful option is the *same-interface* keyword, which instructs the router to allow fast switching of packets that come in and go back out through the same physical interface:

```
Router(config)#interface Serial0/0
Router(config-if)#ip route-cache same-interface
```

You should use this option when the router frequently needs to switch packets between different networks that all connect to the same port. This could be the case for Frame Relay networks, as well as for LANs that use subinterfaces or secondary IP addresses.

Cisco supplies three useful commands to look at CEF performance. The first is *show cef interface*:

```
Router#show cef interface FastEthernet0/0
FastEthernet0/1 is up (if_number 4)
  Corresponding hwidb fast_if_number 4
  Corresponding hwidb firstsw->if_number 4
  Internet address is 172.22.1.3/24
  ICMP redirects are always sent
  Per packet load-sharing is disabled
  IP unicast RPF check is disabled
  Inbound access list is 120
  Outbound access list is not set
  IP policy routing is disabled
  Hardware idb is FastEthernet0/1
  Fast switching type 1, interface type 18
  IP CEF switching enabled
  IP CEF Feature Fast switching turbo vector
  Input fast flags 0x0, Output fast flags 0x0
  ifindex 4(4)
  Slot 0 Slot unit 1 VC -1
  Transmit limit accumulator 0x0 (0x0)
  IP MTU 1500
Router#
```

The output of this command shows that CEF is enabled on the interface `FastEthernet0/0` as well as information about inbound and outbound ACLs and policies. In this example, you can see that the interface has an access group configured to use access list number 120 to filter inbound traffic.

You can use the *show cef drop* and *show cef not-cef-switched* commands to see more detailed CEF forwarding statistics:

```
Router#show cef drop
CEF Drop Statistics
Slot  Encap_fail  Unresolved Unsupported    No_route       No_adj  ChkSum_Err
RP            71           0           0         105            0           0
Router#show cef not-cef-switched
CEF Packets passed on to next switching layer
Slot  No_adj No_encap Unsupp'ted Redirect  Receive  Options   Access     Frag
RP         0       0           0        0      572        0        0        0
```

These commands show you details of CEF's operation on the router. The first command shows how many packets CEF has had to drop, and the reasons for the drops. The *Slot* column in the output of both commands refers to the VIP slot where the packets were received. In this case, the router didn't have any VIP cards because it was a Cisco2600. So all packets are received by the route processor, which is indicated by the RP in the leftmost column.

The *Encap_fail* column in the *show cef drop* output shows the number of packets that CEF has dropped because they were incomplete and there was no adjacency route in the CEF table. *Unresolved* indicates the number of packets dropped because CEF could not resolve the destination address prefix. If there had been any packets that could not be switched by CEF because of unsupported features, they would appear in the *Unsupported* column. The *No_route* column shows the number of packets dropped because CEF didn't have a route to the destination. Similarly, *No_adj* shows the number of packets for which CEF did not have an entry in its adjacency table, so it had to send an ARP query. Finally, *ChkSum_Err* shows the number of times that CEF had to drop packets because they were corrupted.

The *show cef not-cef-switched* command has similar output. *No_adj* is the same here as it was in the *show cef drop* command, while *Unsupp'ted* is the same as the *Unsupported* column. The *No_encap* column counts the number of packets that could not be switched because they were encapsulated in another protocol. *Redirect* means that CEF has had to send these packets to another algorithm, usually process switching, to handle. And *Receive* lists the number of packets that were received from another internal switching algorithm. The remaining columns are rarely of interest in practice.

You can display the CEF version of the routing table with the *show ip cef* command:

```
Router#show ip cef
Prefix              Next Hop            Interface
0.0.0.0/0           172.25.1.1          FastEthernet0/0.1
0.0.0.0/32          receive
172.16.2.0/24       attached            FastEthernet0/1
                    attached            FastEthernet1/1
172.22.1.0/24       attached            FastEthernet0/1
172.22.1.0/32       receive
172.22.1.3/32       receive
172.22.1.4/32       172.22.1.4          FastEthernet0/1
<many lines deleted>
Router#
```

Notice in this output that there are actually two equal-cost routes to `172.16.2.0/24`. CEF supports

load balancing between these two paths.

You can expand the detail on these entries with the *show ip cef detail* command:

```
Router#show ip cef detail
IP CEF with switching (Table Version 31), flags=0x0
  31 routes, 0 reresolve, 0 unresolved (0 old, 0 new), peak 1
  31 leaves, 21 nodes, 25560 bytes, 62 inserts, 31 invalidations
  0 load sharing elements, 0 bytes, 0 references
  universal per-destination load sharing algorithm, id 0697166A
  3(1) CEF resets, 0 revisions of existing leaves
  Resolution Timer: Exponential (currently 1s, peak 1s)
  0 in-place/0 aborted modifications
  refcounts:  5672 leaf, 5632 node

Adjacency Table has 5 adjacencies
0.0.0.0/0, version 27, cached adjacency 172.25.1.1
0 packets, 0 bytes
  via 172.25.1.1, FastEthernet0/0.1, 0 dependencies
    next hop 172.25.1.1, FastEthernet0/0.1
    valid cached adjacency
0.0.0.0/32, version 0, receive
172.16.2.0/24, version 21, attached, connected
0 packets, 0 bytes
  via FastEthernet0/0.2, 0 dependencies
    valid glean adjacency
172.16.2.0/32, version 10, receive
172.16.2.1/32, version 9, receive
172.16.2.255/32, version 11, receive
172.22.1.0/24, version 22, attached, connected
0 packets, 0 bytes
  via FastEthernet0/1, 0 dependencies
    valid glean adjacency
172.22.1.0/32, version 16, receive
<many lines deleted>
Router#
```

Top

Recipe 11.2 Setting the DSCP or TOS Field

11.2.1 Problem

You want the router to mark the DSCP or TOS field of an IP packet to affect its priority through the network.

11.2.2 Solution

The solution to this problem depends on the sort of traffic distinctions you want to make, as well the version of IOS you are running in your routers.

There must be something that defines the different types of traffic that you wish to prioritize. In general, the simpler the distinctions are to make, the better. This is because all of the tests take router resources and introduce processing delays. The most common rules for distinguishing between traffic types use the packet's input interface and simple IP header information such as TCP port numbers. The following examples show how to set an IP Precedence value of immediate (2) for all FTP control traffic that arrives through the `serial0/0` interface, and an IP Precedence of priority (1) for all FTP data traffic. This distinction is possible because FTP control traffic uses TCP port 21, and FTP data uses port 20.

The new method for configuring this uses class maps. Cisco first introduced this feature in IOS Version 12.0(5)T. This method first defines a class-map that specifies how the router will identify this type of traffic. It then defines a policy-map that actually makes the changes to the packet's TOS field:

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#access list 101 permit any eq ftp any
Router(config)#access list 101 permit any any eq ftp
Router(config)#access list 102 permit any eq ftp-data any
Router(config)#access list 102 permit any any eq ftp-data
Router(config)#class-map match-all ser00-ftpcontrol
Router(config-cmap)#description branch ftp control traffic
Router(config-cmap)#match input-interface serial0/0
Router(config-cmap)#match access-group 101
Router(config-cmap)#exit
Router(config)#class-map match-all ser00-ftpdata
Router(config-cmap)#description branch ftp data traffic
Router(config-cmap)#match input-interface serial0/0
Router(config-cmap)#match access-group 102
Router(config-cmap)#exit
Router(config)#policy-map serialftppolicy
```

```
Router(config-pmap)#description branch ftp traffic policy
Router(config-pmap)#class ser00-ftpcontrol
Router(config-pmap-c)#set ip precedence immediate
Router(config-pmap-c)#exit
Router(config-pmap)#class ser00-ftpdata
Router(config-pmap-c)#set ip precedence priority
Router(config-pmap-c)#exit
Router(config-pmap)#exit
Router(config)#interface serial0/0
Router(config-if)#ip route-cache policy
Router(config-if)#service-policy input serialftppolicy
Router(config-if)#end
Router#
```

For earlier IOS versions, where class maps were not available, you have to use policy-based routing to alter the TOS field in a packet. Applying this policy to the interface tells the router to use this policy to test all incoming packets on this interface and rewrite the ones that match the route map:

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#access list 101 permit any eq ftp any
Router(config)#access list 101 permit any any eq ftp
Router(config)#access list 102 permit any eq ftp-data any
Router(config)#access list 102 permit any any eq ftp-data
Router(config)#route-map serialftp-rtmap permit 10
Router(config-route-map)#match ip address 101
Router(config-route-map)#set ip precedence immediate
Router(config-route-map)#exit
Router(config)#route-map serialftp-rtmap permit 20
Router(config-route-map)#match ip address 102
Router(config-route-map)#set ip precedence priority
Router(config-route-map)#exit
Router(config)#interface serial0/0
Router(config-if)#ip policy route-map serialftp-rtmap
Router(config-if)#ip route-cache policy
Router(config-if)#end
Router#
```

11.2.3 Discussion

Before you can tag a packet for special treatment, you have to have an extremely clear idea of what types of traffic need special treatment, as well as precisely what sort of special treatment they will need. In the example, we have decided to give special priority to FTP traffic received on a specific serial interface. We show how to do this using both the old and new configuration techniques.

This may appear to be a somewhat artificial example. After all, why would you care about tagging inbound traffic that you have already received from a low-speed interface? Actually, one of the most important principles for implementing QoS in a network is that you should always tag the packet as early as possible, preferably at the edges of the network. Then, as it passes through the network, each

router only needs to look at the tag, and doesn't need to do any additional classification. In this case, we would ensure that the FTP traffic returning in the other direction is tagged by the first router that receives it. So the outbound traffic has already been tagged, and it is a waste of router resources to reclassify the outbound packets.

Many organizations actually take this idea of marking at the edges one step further, and remark every received packet. This helps to ensure that users aren't requesting special QoS privileges that they aren't allowed to have. However, you should be careful of this because it can sometimes disrupt legitimate markings. For example, a real-time application might use RSVP to reserve bandwidth through the network. It is important that the packets for this application have the appropriate Expedited Forwarding (EF) DSCP marking, or the network might not handle them properly. However, you also don't want to let other non-real-time applications from this same source have the same EF priority level. So, if you are going to configure your routers to remark all incoming packets at the edges, make sure you understand what incoming markings are legitimate.

Recipe 15.9 shows another interesting variation of this idea. In that case, the routers are running DLSw to bridge SNA traffic through an IP network. So the routers themselves actually create the IP packets. This creates an additional challenge because there is no incoming interface, so that recipe uses local policy-based routing. The fact that the router creates the packets also gives it an important advantage, because it doesn't have to consider any DLSw packets that might just happen to pass through.

The advantages of the newer class map method aren't obvious in this example, but one of the first big advantages appears if you want to use the more modern DSCP tagging scheme. Because the older policy-based routing method doesn't directly support DSCP, you have to fake it by setting both the IP Precedence and the TOS separately as follows:

```
Router(config)#route-map serialftp-rtmap permit 10
Router(config-route-map)#match ip address 115
Router(config-route-map)#set ip precedence immediate
Router(config-route-map)#set ip tos max-throughput
```

In this case, the packet will wind up with an IP Precedence value of immediate, or 2 (`10` in binary), and TOS of max-throughput or 4 (`0100` in binary). Combining the bit patterns gives you `0100100`, but, as we discuss in Appendix B, DSCP only uses the first 6 bits,`010010`. If you look up this bit combination in Table B.3 in Appendix B, you will see that it corresponds to a value of AF21, which is Class 2 and lowest drop precedence.

Doing the same thing with the class map method is much more direct:

```
Router(config)#policy-map serialftppolicy
Router(config-pmap)#class serialftpclass
Router(config-pmap-c)#set ip dscp af21
```

Class maps will also be useful later in this chapter when we talk about class-based Weighted Fair Queueing and class-based traffic shaping.

It is important to note that throughout this entire example, we have only put a special value into the packet's TOS or DSCP field. This, by itself, doesn't affect how the packet is forwarded through the network. To forward properly you must ensure that as each router in the network forwards these marked packets, the interface queues will react appropriately to this information.

Finally, while this recipe shows two useful ways of marking packets, Recipe 11.12 shows still another method, using Committed Access Rate (CAR) features. CAR tends to be more efficient on higher speed interfaces.

11.2.4 See Also

Recipe 11.3; Recipe 11.5; Recipe 11.6; Recipe 11.7; Recipe 11.12; Recipe 11.13

Top

Recipe 11.3 Using Priority Queueing

11.3.1 Problem

You want to enable strict priority queues on an interface so that the router always handles high priority packets first.

11.3.2 Solution

To enable Priority Queueing on an interface, you must first define the priority list, and then apply it to the interface:

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#access list 101 permit ip any any precedence 5 tos 12
Router(config)#access list 102 permit ip any any precedence 4
Router(config)#access list 103 permit ip any any precedence 3
Router(config)#priority-list 1 protocol ip high list 101
Router(config)#priority-list 1 protocol ip medium list 102
Router(config)#priority-list 1 protocol ip normal list 103
Router(config)#priority-list 1 default low
Router(config)#interface Ethernet0
Router(config-if)#priority-group 1
Router(config-if)#end
Router#
```

11.3.3 Discussion

As we discuss in Appendix B, priority queues strictly ensure that high priority packets are always handled before lower priority packets. We stress that using pure Priority Queueing like this is usually a bad idea because the higher priority traffic can take all of the available bandwidth and completely starve all other network traffic. You want to use this style of queueing only when you can be absolutely certain that the aggregate bandwidth of all high priority traffic will never consume the available link bandwidth. This could be the case, for example, if the high priority traffic is shaped before reaching this router, or for applications like Voice over IP (VoIP) that use a relatively constant amount of bandwidth, and don't burst above this constant rate.

The *priority-list* command has a relatively flexible syntax for identifying what types of traffic will use which queues. However, we prefer the access list method shown in the example. This is because it gives the greatest range of possibilities for identifying traffic types.

In the example, we use access list 101 to decide which packets to send to the high priority queue:

```
Router(config)#access list 101 permit ip any any precedence 5 tos 12
```

If you write out the bit patterns for an IP Precedence value of 5 and a TOS of 12, you get `101` and `1100`. Combining these together and dropping the last bit gives `101110`, which is identical to the EF DSCP value. This is typically the DSCP value that is used to mark packets for real-time applications.

Cisco introduced a *dscp* keyword to the access list command in IOS Version 12.1(5)T. This allows you to accomplish the same thing with a slightly simpler access list. This access list should also process faster because it only makes one comparison instead of two:

```
Router(config)#access list 101 permit ip any any dscp ef
```

The access lists that define the other queues also select specific IP Precedence values. This is because we want to carefully limit the amount of processing that the router has to do. The less the access list has to look at, the better.

Note also that the router will process the priority list in the order that it was entered. In general you will want to keep queueing latency for high priority packets as low as possible. This is why we define the higher priority queues first.

In the example, we also specifically included a command to put any unmatched packets into the low priority queue:

```
Router(config)#priority-list 1 default low
```

If we had not included this command, the router would have used the normal priority queue for any unmatched packets by default.

You can look at Priority Queueing information on an interface with the *show interface* command:

```
Router#show interface Ethernet0
Ethernet0 is up, line protocol is up
  Hardware is Lance, address is 0000.0cf0.8460 (bia 0000.0cf0.8460)
  Internet address is 192.168.1.201/24
MTU 1500 bytes, BW 10000 Kbit, DLY 1000 usec,
   reliability 255/255, txload 1/255, rxload 1/255
Encapsulation ARPA, loopback not set, keepalive set (10 sec)
ARP type: ARPA, ARP Timeout 04:00:00
Last input 00:00:00, output 00:00:00, output hang never
Last clearing of "show interface" counters never
Input queue: 0/75/0 (size/max/drops); Total output drops: 0
Queueing strategy: priority-list 1
Output queue (queue priority: size/max/drops):
   high: 0/20/0, medium: 0/40/0, normal 0/60/0, low 0/80/0
5 minute input rate 1000 bits/sec, 2 packets/sec
5 minute output rate 2000 bits/sec, 2 packets/sec
```

```
     7390 packets input, 655552 bytes, 0 no buffer
     Received 6687 broadcasts, 0 runts, 0 giants, 0 throttles
    0 input errors, 0 CRC, 0 frame, 0 overrun, 0 ignored, 0 abort
     0 input packets with dribble condition detected
     81097 packets output, 6240100 bytes, 0 underruns
     2 output errors, 0 collisions, 7 interface resets
    0 babbles, 0 late collision, 0 deferred
    2 lost carrier, 0 no carrier
     0 output buffer failures, 0 output buffers swapped out
Router#
```

In this case, you can see that the high priority queue has a maximum depth of 20 packets. The medium queue can hold 40 packets, normal holds 60, and the low priority queue can hold 80 packets. This increasing queue depth pattern is necessary to help deal with queue starvation problems. You can modify these default values as follows:

```
Router(config)#priority-list 1 queue-limit 10 15 25 35
```

This command sets the depths for all of the queues in increasing order. This particular example would set the high priority queue to hold a maximum of 10 packets, 15 for the medium queue, 25 for the normal queue, and 35 for the low priority queue.

Note that the router will automatically use the high priority queue for critical network control information such as routing updates and keepalives. If these packets are not sent in a timely fashion, it can disrupt how the network functions. If the router were to put this critical information into a lower priority queue, there would be a danger that higher priority application traffic could starve the lower priority queues, and disrupt routing or possibly even bring down parts of the network.CBWFQ and Cisco's new Low Latency Queueing (LLQ) algorithm offer all of the advantages of Priority Queueing discussed here, and fewer of the disadvantages. This feature is discussed in Recipe 11.13. We recommend using LLQ instead of Priority Queueing if your router supports it. Cisco introduced LLQ in IOS level 12.0(6)T.

11.3.4 See Also

Recipe 11.13

Top

Recipe 11.4 Using Custom Queueing

11.4.1 Problem

You want to configure custom queueing on an interface to give different traffic streams a share of the bandwidth according to their IP Precedence levels.

11.4.2 Solution

Implementing Custom Queueing on a router is a two-step procedure. First you must define the traffic types that will populate your queues. And then you apply the queueing method to an interface:

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#access list 103 permit ip any any precedence 5
Router(config)#access list 104 permit ip any any precedence 4
Router(config)#access list 105 permit ip any any precedence 3
Router(config)#access list 106 permit ip any any precedence 2
Router(config)#access list 107 permit ip any any precedence 1
Router(config)#queue-list 1 protocol ip  3 list 103
Router(config)#queue-list 1 protocol ip  4 list 104
Router(config)#queue-list 1 protocol ip  5 list 105
Router(config)#queue-list 1 queue 5 byte-count 3000 limit 55
Router(config)#queue-list 1 protocol ip  6 list 106
Router(config)#queue-list 1 protocol ip  7 list 107
Router(config)#queue-list 1 default 8
Router(config)#interface HSSI0/0
Router(config-if)#custom-queue-list 1
Router(config-if)#end
Router#
```

11.4.3 Discussion

When you enable Custom Queueing, the router automatically creates 16 queues for application traffic plus one more for system requirements. You can look at the queues with a normal *show interface* command:

```
Router#show interface Ethernet0
Ethernet0 is up, line protocol is up
  Hardware is Lance, address is 0000.0cf0.8460 (bia 0000.0cf0.8460)
  Internet address is 192.168.1.201/24
  MTU 1500 bytes, BW 10000 Kbit, DLY 1000 usec,
```

```
    reliability 255/255, txload 2/255, rxload 1/255
Encapsulation ARPA, loopback not set, keepalive set (10 sec)
ARP type: ARPA, ARP Timeout 04:00:00
Last input 00:00:00, output 00:00:00, output hang never
Last clearing of "show interface" counters never
Input queue: 2/75/0 (size/max/drops); Total output drops: 0
Queueing strategy: custom-list 1
Output queues: (queue #: size/max/drops)
   0: 0/20/0 1: 0/20/0 2: 0/20/0 3: 0/20/0 4: 0/20/0
   5: 0/55/3 6: 5/20/0 7: 0/20/0 8: 0/20/0 9: 0/20/0
   10: 0/20/0 11: 0/20/0 12: 0/20/0 13: 0/20/0 14: 0/20/0
   15: 0/20/0 16: 0/20/0
5 minute input rate 5000 bits/sec, 12 packets/sec
5 minute output rate 106000 bits/sec, 24 packets/sec
    132910 packets input, 14513345 bytes, 0 no buffer
    Received 109570 broadcasts, 0 runts, 0 giants, 0 throttles
   9 input errors, 0 CRC, 0 frame, 0 overrun, 9 ignored, 0 abort
    0 input packets with dribble condition detected
    1028116 packets output, 85603681 bytes, 0 underruns
    1 output errors, 42 collisions, 8 interface resets
   0 babbles, 0 late collision, 4 deferred
   1 lost carrier, 0 no carrier
    0 output buffer failures, 0 output buffers swapped out
Router#
```

In this output you can see that queue number 6 currently has five packets queued and waiting for delivery (6: 5/20/0), while queue number 5 has had to drop three packets due to congestion (5: 0/55/3).

The example assigns queue number 3 for all packets with the highest application IP Precedence value of 5. Similarly, packets with Precedence 4 use queue number 4, Precedence 3 use queue 5, Precedence 2 use queue 6, Precedence 1 use queue 7, and everything else uses queue number 8.

Custom Queueing does not assign a default queue for unclassified traffic, so you must remember to do this. The command in the example defines the default as queue number 8:

```
Router(config)#queue-list 1 default 8
```

Note that if there is another non-IP protocol such as IPX configured on this interface, it will also use the default queue. If you prefer to give this other protocol its own set of queues, you can use define them using access lists for that protocol. The configuration is nearly identical to the IP example we have shown, except for the exact access list syntax, which naturally depends on the protocol.

By default, the Custom Queueing scheduler visits all queues in order and takes an average of 1500 bytes from each, and each queue can hold up to 20 packets. In the example, we changed these default values for queue number 5:

```
Router(config)#queue-list 1 queue 5 byte-count 3000 limit 55
```

This tells the scheduler to take an average of 3000 bytes from this queue on each pass, and to store up to

55 packets in the queue. Increasing the number of bytes will effectively increase the share of the bandwidth that this queue receives. Increasing the queue depth decreases the probability of tail drops. But it also increases the amount of time that a packet could theoretically spend in the queue, which may increase latency and jitter.

In this example, all of the traffic types are selected by the IP Precedence value. It is also possible to select based on specific applications. You can do this either with an access list or, in some cases, using keywords in the *queue-list* command. For example, if you wanted to select all DLSw traffic and send it to queue number 9, you could create an access list:

```
Router(config)#access list 117 permit ip any eq 2065 any
Router(config)#access list 117 permit ip any any eq 2065
Router(config)#access list 117 permit ip any eq 2067 any
Router(config)#access list 117 permit ip any any eq 2067
Router(config)#queue-list 1 protocol ip  9 list 117
```

Or you could do it like this:

```
Router(config)#queue-list 1 protocol dlsw  9
```

This second method is clearly easier, but the number of protocol types that can be defined this way is unfortunately rather limited.

We have three important final notes on Custom Queueing that you should bear in mind. The first point is that if traffic from all of these streams is present, the router will share traffic between them. In this example, we have used six different queues: one for each of the five application precedence levels plus a default. By default, each will receive a roughly equal share of the total bandwidth. So you may be surprised to find that, despite imposing different queues for the different traffic-only types, the important traffic still doesn't get a large enough share of the bandwidth. You can affect this with the *byte-count* keyword, as we discussed earlier. Note that the queues are serviced by byte count rather than packet count. So suppose you have two queues, one of which supports an interactive session with many short packets, while the other contains a bulk transfer with a few large packets. If you configure the router to service these queues with the same byte-count, it will tend to forward a lot more of the small packets. But the net share of the bandwidth will be roughly equal on average.

Secondly, in Custom Queueing, the traffic within each queue competes directly with all other traffic in the same queue. So, for example, if one user sends a burst of application traffic that fills one of the queues, this will cause tail drops for other users whose traffic uses the same queue. This will cause a smaller version of the global problem of a FIFO queue that we discuss in Appendix B.

The third point is that the more queues you define, the smaller the share of the total bandwidth each queue receives. Further, having more queues increases the amount of processing the router has to do to segregate the traffic.

The second and third points compete with one another. The second one tends to point toward

increasing the number of queues to limit the competition within each queue. But the third point should convince you that there is a point of diminishing returns where more queues will not help the situation. In practice, the third rule tends to win out. It rarely turns out to be beneficial to have more than five or six Custom Queues unless some of those queues are only used very lightly.

Custom Queueing is an older QoS mechanism on Cisco routers. In most cases, you will likely find that a newer algorithm such as CBWFQ will be more flexible and give better results.

Top

Recipe 11.5 Using Custom Queues with Priority Queues

11.5.1 Problem

You want to combine Custom Queueing with Priority Queueing on an interface so the highest priority packets are always handled first, and lower priority traffic streams share bandwidth with one another.

11.5.2 Solution

You can split the queues so that some use Priority Queueing and the remainder use Custom Queueing:

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#access list 101 permit ip any any precedence 7
Router(config)#access list 102 permit ip any any precedence 6
Router(config)#access list 103 permit ip any any precedence 5
Router(config)#access list 104 permit ip any any precedence 4
Router(config)#access list 105 permit ip any any precedence 3
Router(config)#access list 106 permit ip any any precedence 2
Router(config)#access list 107 permit ip any any precedence 1
Router(config)#queue-list 1 protocol ip  1 list 101
Router(config)#queue-list 1 protocol ip  2 list 102
Router(config)#queue-list 1 protocol ip  3 list 103
Router(config)#queue-list 1 protocol ip  4 list 104
Router(config)#queue-list 1 protocol ip  5 list 105
Router(config)#queue-list 1 protocol ip  6 list 106
Router(config)#queue-list 1 protocol ip  7 list 107
Router(config)#queue-list 1 lowest-custom  4
Router(config)#interface HSSI0/0
Router(config-if)#custom-queue-list 1
Router(config-if)#end
Router#
```

11.5.3 Discussion

This example is similar to Recipe 11.4, which looked at a pure Custom Queueing example. In this case, however, we have added the command:

```
Router(config)#queue-list 1 lowest-custom  4
```

This command allows you to mix Custom and Priority Queue types. Note that this command only works with queue-list number 1. It is not available for any other queue lists.

In this example, queue number 4 is the lowest numbered Custom Queue. So, in this example, queues 1, 2 and 3 are all Priority Queues. This means that the router will deliver all of the packets in queue number 1, then all of the packets in queue number 2, then all of the packets in queue number 3. And then, if these high priority queues are all empty, it will use custom queueing to deliver the packets in the lower priority queues.

The main advantage to this sort of configuration is that it gives absolute priority to real-time applications. This is important not because of the bandwidth, but because Priority Queueing the real-time applications minimizes their queueing latency. However, as with the pure Priority Queueing example in Recipe 11.3, you have to be extremely careful to prevent the high priority traffic from starving the other queues.

11.5.4 See Also

Recipe 11.3; Recipe 11.4

Top

Recipe 11.6 Using Weighted Fair Queueing

11.6.1 Problem

You want your routers to use the TOS/DSCP fields when forwarding packets.

11.6.2 Solution

The simplest way to make your routers use DSCP or TOS information is to just make sure that Weighted Fair Queueing (WFQ) is enabled:

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#interface Serial0/0
Router(config-if)#fair-queue
Router(config-if)#end
Router#
```

WFQ is enabled by default on all interfaces of E1 speeds (roughly 2Mbps) or less. You can enable WFQ on higher speed interfaces as well, but we don't recommend it.

To configure more specific behavior, you can tell WFQ how to allocate its queues:

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#interface Serial0/0
Router(config-if)#fair-queue 64 512 10
Router(config-if)#end
Router#
```

11.6.3 Discussion

Before we discuss these examples in any detail, we should mention that WFQ works well even if you don't use any TOS/DSCP marking. In this case, it simply gives the same default weighting to every flow, which is the same as conventional Fair Queueing (without the weights). As we discuss in Appendix B, Fair Queueing is much more effective than FIFO queueing. However, with TOS/DSCP marking, WFQ really shows its value, giving higher priority flows more of the bandwidth, and preventing high volume flows from starving low volume flows regardless of their TOS markings.

The first example just enables WFQ on the interface. In fact, this is the default for all interfaces that

operate slower than 2.048Mbps (E1 speed), except for interfaces that use LAPB or SDLC. WFQ does not work with LAPB or SDLC.

The second example is a little bit more interesting, though, because it changes the default queues. The *fair-queue* command has three optional parameters. In the example, we specified:

```
Router(config-if)#fair-queue 64 512 10
```

The first number specifies the congestive discard threshold. This just means that if there are more than 64 packets in any given queue, the router will start to discard any new packets. The default threshold is 64.

The second number is the number of dynamic queues. The values must be a power of 2 multiplied of 16 to a maximum of 4096 (i.e., one of the following values: 16, 32, 64, 128, 256, 512, 1024, 2048, or 4096). The default value is 256. If this interface must support a large number of flows, then it is a good idea to choose a larger value.

The last number (10 in this case) is the number of queues that the router will set aside for RSVP reservation requests. Please see Recipe 11.9 for more information about RSVP.

In most cases, the default parameters are good enough. However, there are two times in particular when it is useful to modify them. First, if the interface must support an extremely large number of flows, you will get better performance by using a larger number of queues. However, be careful doing this on faster interfaces because the router may start to have trouble processing the additional queues. In that case, you could actually get a performance improvement by decreasing the number of queues. In this case, each queue could wind up simultaneously handling a few distinct flows.

You will also need to change the default queue parameters if you are using RSVP to reserve queues. By default, this parameter is zero. If you are using RSVP with WFQ, you must allocate some reserved queuing space.

11.6.4 See Also

Recipe 11.9

Top

Recipe 11.7 Using Class-Based Weighted Fair Queueing

11.7.1 Problem

You want to use Class-Based Weighted Fair Queueing on an interface.

11.7.2 Solution

There are three steps to configuring Class-Based Weighted Fair Queueing (CBWFQ) on a router. First, you have to create one or more class maps that describe the traffic types. Then you create a policy map that tells the router what to do with these traffic types. Finally you need to attach this policy map to one or more of the router's interfaces:

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#class-map highprec
Router(config-cmap)#description Highest priority Prec=5
Router(config-cmap)#match ip precedence 5
Router(config-cmap)#exit
Router(config)#class-map medhiprec
Router(config-cmap)#description Medium-high priority Prec=4
Router(config-cmap)#match ip precedence 4
Router(config-cmap)#exit
Router(config)#class-map medloprec
Router(config-cmap)#description Medium-low priority Prec=2,3
Router(config-cmap)#match ip precedence 2 3
Router(config-cmap)#exit
Router(config)#policy-map cbwfqpolicy
Router(config-pmap)#class highprec
Router(config-pmap-c)#bandwidth percent 25
Router(config-pmap-c)#exit
Router(config-pmap)#class medhiprec
Router(config-pmap-c)#bandwidth percent 25
Router(config-pmap-c)#exit
Router(config-pmap)#class medloprec
Router(config-pmap-c)#bandwidth percent 25
Router(config-pmap-c)#exit
Router(config-pmap)#class class-default
Router(config-pmap-c)#fair-queue 512
Router(config-pmap-c)#queue-limit 96
Router(config-pmap-c)#exit
Router(config-pmap)#exit
Router(config)#interface serial0/1
```

```
Router(config-if)#service-policy output cbwfqpolicy
Router(config-if)#end
Router#
```

This feature is available in IOS levels 12.0(5)T and higher.

11.7.3 Discussion

CBWFQ need not be significantly different from regular WFQ. In the example we have defined all traffic with an IP Precedence value of critical (5) to have a special queue. We have also created a single queue for traffic with Precedence 4, and another one for traffic with Precedence values of 2 and 3. All other traffic, including traffic with Precedence 0 and 1 as well as all non-IP traffic, uses regular WFQ. To make this fact slightly more clear, we have modified the default WFQ parameters with the following commands:

```
Router(config)#policy-map cbwfqpolicy
Router(config-pmap)#class class-default
Router(config-pmap-c)#fair-queue 512
Router(config-pmap-c)#queue-limit 96
```

This simply modifies the default WFQ behavior for all traffic that doesn't match one of the other defined classes. It sets the number of WFQ queues to 512, and sets the queue depth to a maximum of 96 packets. You could achieve the same effect using the *fair-queue* interface command from Recipe 11.6:

```
Router(config-if)#fair-queue 96 512 0
```

But that example doesn't give you the ability to also have separate queues for special classes of traffic, as shown in this recipe. The final argument for this *fair-queue* interface command specifies the number of queues to set aside for RSVP. We are trying to duplicate the effect of the cbwfqpolicy policy map, which doesn't include any RSVP queues, so we have set the last argument to zero here. Please refer to Recipe 11.6 for more information on this command.

You can create up to 64 class-based queues for use with CBWFQ. You can control the share of the bandwidth to each queue using the *bandwidth* keyword either using an absolute value in kilobits per second, or a percentage of the total available bandwidth. The following example shows the syntax for using a percentage:

```
Router(config-pmap)#class highprec
Router(config-pmap-c)#bandwidth percent 25
Router(config-pmap-c)#exit
```

The *bandwidth percent* command is available in IOS levels 12.1(1) and higher. For earlier releases, you can only specify an absolute bandwidth:

```
Router(config-pmap-c)#bandwidth 5000
```

The argument for this version of the command is a value in Kbps between 8 and 2,000,000, which should be sufficient for most interface types. Note that the upper limit here is 2Mbps, which is roughly the E1 speed mentioned earlier as the effective upper limit to using WFQ. Because CBWFQ generally uses fewer queues and doesn't need to sort based on flow, you can use it for higher speed interfaces as well. However, you should let your average CPU utilization be your guide. If you do too many tests when classifying packets, you might find that the router can't keep up with high packet rates.

In both versions you have to keep two important factors in mind. First, although this is essentially a Layer 3 feature, you have to include any Layer 2 framing overhead when configuring the bandwidth. If a given queue supports a streaming multimedia application with a known bit rate, it is often a good idea to slightly overestimate the requirements to include this Layer 2 overhead. If the application doesn't use the excess, CBWFQ allocates it to other queues.

The second important factor is that the total allocated bandwidth must not exceed a configurable maximum value. By default, this maximum is 75%. You can change it (for example, to 80%) by using the following interface level command:

```
Router(config-if)#max-reserved-bandwidth 80
```

You would apply this command to the interface that runs CBWFQ and needs a little extra reserved capacity. It is usually best to leave this at its default value, however. The router uses the remainder for unclassified traffic and network control packets. In this case, we have configured WFQ for the unclassified traffic. It is vital, however, to reserve enough bandwidth for critical network functions such as Layer 2 keepalive frames and routing protocols.

Creating the policy map alone doesn't actually change the way the router behaves. To do that you have to attach this policy to an interface as follows:

```
Router(config)#interface serial0/1
Router(config-if)#service-policy output cbwfqpolicy
```

One of the classes defined in this example is a high priority class that we called "*highpriority*". This class map simply looks for traffic that is tagged with an IP Precedence value of 5. The policy map then tells the interface to give up to 25% of its bandwidth to this high priority traffic. If there is not enough high priority traffic to use this, the router will allocate the excess to the remaining traffic.

We will expand on this concept in Recipe 11.13.

11.7.4 See Also

Recipe 11.2; Recipe 11.6; Recipe 11.13

Top

Recipe 11.8 Controlling Congestion with WRED

11.8.1 Problem

You want to control congestion on an interface before it becomes a problem.

11.8.2 Solution

The syntax for configuring WRED changed with the introduction of class-based QoS. The old method defined WRED across an entire interface:

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#interface HSSI0/0
Router(config-if)#random-detect
Router(config-if)#random-detect precedence 0 10 20 10
Router(config-if)#random-detect precedence 1 12 20 10
Router(config-if)#random-detect precedence 2 15 25 15
Router(config-if)#random-detect precedence 3 18 25 15
Router(config-if)#random-detect precedence 4 20 30 20
Router(config-if)#random-detect precedence 5 22 30 20
Router(config-if)#random-detect precedence 6 30 40 25
Router(config-if)#random-detect precedence 7 40 50 100
Router(config-if)#random-detect precedence RSVP 45 50 100
Router(config-if)#end
Router#
```

The new configuration method uses the same syntax as CBWFQ:

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#class-map Prec5
Router(config-cmap)#description Critical
Router(config-cmap)#match ip precedence 5
Router(config-cmap)#exit
Router(config)#policy-map cb_wred
Router(config-pmap)#class Prec5
Router(config-pmap-c)#random-detect dscp-based
Router(config-pmap-c)#exit
Router(config-pmap)#class class-default
Router(config-pmap-c)#fair-queue 512
Router(config-pmap-c)#queue-limit 96
Router(config-pmap-c)#random-detect dscp-based
```

```
Router(config-pmap-c)#exit
Router(config-pmap)#exit
Router(config)#interface HSSI0/1
Router(config-if)#service-policy output cb_wred
Router(config-if)#end
Router#
```

11.8.3 Discussion

For the older method, you can set up the drop probabilities according to IP Precedence values using the following command:

```
Router(config-if)#random-detect precedence 7 40 50 100
```

The first argument after the *precedence* keyword here is the IP Precedence value. The options are any integer between and 7 or the keyword *RSVP*. After this are the minimum threshold, maximum threshold, and the so-called *mark probability* denominator.

The minimum threshold is the number of packets that must be in the queue before the router starts to discard. The probability at the minimum threshold is essentially zero, but it rises linearly as the number of packets in the queue rises. The maximum probability occurs at the maximum threshold. You specify the actual value of the probability at this maximum using the mark probability denominator. In this case, we have set the value to 100, which means that at the maximum, we will discard 1 packet in 100. Therefore, halfway between the maximum and minimum thresholds, the router will drop 1 packet in 200.

As we discuss in Appendix B, the router doesn't necessarily drop packets when the queue depth reaches the minimum threshold. Rather, it uses a moving average so that temporary bursts of data are not dropped. This configured minimum is the lower limit of this moving average, which is reached only when the congestion continues for a longer period of time.

If you do not change these values, the defaults take IP Precedence values into account. The default mark probability denominator is 10, so the router will discard at most 1 packet in 10. The default maximum threshold depends on the speed of the interface and the router's capacity for buffering packets, but it is the same for all Precedence values. So, by default, the only differences between WRED's treatment of different IP Precedence levels is in the minimum threshold. The default minimum threshold for packets with an IP Precedence of 0 is 50% of the maximum threshold. This value rises linearly with Precedence so that the minimum threshold for Precedence 7 and packets with RSVP reserved-bandwidth allocations are almost the same as the maximum threshold.

In the new-style example, we have only created one class-based queue to show the principle. In practice, you would probably want to create more than this. All of the traffic that doesn't have an IP Precedence value of 5 uses the default queue, where we have configured both WFQ and WRED.

This example uses DSCP-based random detection. WRED has a built-in ability to discriminate based on DSCP value, so that traffic streams with higher drop precedence values are more likely to drop

packets. The default WRED settings when using DSCP-based random detection are shown in Table 11-1.

Table 11-1. Default parameters for DSCP-based WRED

DSCP value	Minimum thresholdqueue depth	Maximum thresholdqueue depth	Drop probabilityat maximum
AFx1	32	40	1/10
AFx2	28	40	1/10
AFx3	24	40	1/10

As Table 11-1 shows, the default DSCP-based thresholds are the same for every class. So, for example, AF12, AF22, AF32 and AF42 all begin dropping packets in a sustained congestion situation when the queue depth reaches 28 packets. They reach their maximum drop probability when there are 40 packets in the queue. In all cases, the drop probability at the maximum threshold value is 1/10 (the mark probability), meaning that the router will randomly drop 1 packet in 10.

You can change these values in a policy map as follows:

```
Router(config-pmap)#class AF1x
Router(config-pmap-c)#bandwidth percent 20
Router(config-pmap-c)#random-detect dscp-based
Router(config-pmap-c)#random-detect dscp af13  10 20
Router(config-pmap-c)#random-detect dscp af12  20 50
Router(config-pmap-c)#random-detect dscp af11  50 100 50
Router(config-pmap-c)#exit
```

In each of the *random-detect dscp* commands, the first argument is the DSCP value, followed by the minimum threshold, the maximum threshold, and the denominator of the mark probability. In the case of the AF11 entry, the router will start dropping these packets when there are more than 50 packets in the queue and increase the probability until the number reaches 100. At that point, the probability of dropping a packet of this type will be 1 in 50.

Note that these thresholds apply to all traffic in the queue, not just traffic with this particular DSCP value. So there may be 20 AF11 packets, 10 AF12, and 20 more marked with the AF13 DSCP value. Since this adds up to 50 packets, the router will start to drop the AF11 packets. However, because the maximum thresholds for AF12 and AF13 packets are 50 and 20 respectively, the router will already be dropping packets of these types at the full rate (1 packet in 10 by default) before it starts to drop any AF11 packets.

This example assumes that you want to use DSCP values to control the WRED thresholds. This is not necessary, however. You can also use an unweighted version of the command as follows:

```
Router(config)#class-map AF11
Router(config-cmap)#match ip dscp af11
Router(config-cmap)#exit
Router(config)#policy-map example
Router(config-pmap)#class AF11
Router(config-pmap-c)#bandwidth percent 10
Router(config-pmap-c)#random-detect
Router(config-pmap-c)#exit
```

This is particularly useful when your class definitions already take DSCP values into account, as this class map does. Since there is no variation of DSCP values among the class of packets that have a DSCP value of AF11, it isn't necessary for WRED to look at the DSCP value again.

11.8.4 See Also

Recipe 11.7

Top

Recipe 11.9 Using RSVP

11.9.1 Problem

You want to configure RSVP on your network.

11.9.2 Solution

Basic RSVP configuration is relatively simple. All you need to do is define how much bandwidth can be reserved on the interface:

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#interface FastEthernet0/0
Router(config-if)#ip rsvp bandwidth 128 56
Router(config-if)#end
Router#
```

Some network administrators have to worry about unauthorized use of bandwidth reservation. You can control this by specifying an access list of allowed neighbor devices:

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#access list 15 permit ip 192.168.1.0 0.0.0.255
Router(config)#interface FastEthernet0/0
Router(config-if)#ip rsvp bandwidth 128 56
Router(config-if)#ip rsvp neighbor 15
Router(config-if)#end
Router#
```

11.9.3 Discussion

Note that before you can configure RSVP on an interface, you must first configure the interface for WFQ, CBWFQ, or WRED. This step is not included in this example, to make it easier to focus on the RSVP configuration. For examples of WFQ, CBWFQ, and WRED, please refer to Recipe 11.6, Recipe 11.7, and Recipe 11.8 respectively.

The first example tells the router to pay attention to RSVP signaling, and defines how much bandwidth can be reserved in the following command:

```
Router(config-if)#ip rsvp bandwidth 128 56
```

The first numerical argument, `128`, specifies that applications can reserve a maximum aggregate bandwidth of 128Kbps. The last argument,`56`, means that the largest amount that a single application can request is 56Kbps.

When you use the *ip rsvp neighbor* command, as in the second example, it is important to remember that this router receives RSVP reservation requests from neighboring devices. If this is an access router, then the neighboring device on the local LAN port could be an end device. But for other routers and other interfaces, it is likely that the RSVP request will come from another router, not from the end device making the initial request. So, for router-to-router connections, it may not be useful to specify an access list because all RSVP requests, legitimate or not, will come from a neighboring router. The best place to control which devices are allowed to reserve bandwidth is on the access router.

There are several useful show commands to look at the RSVP configuration of your router as well as the dynamic reservation requests. The first of these is the *show ip rsvp interface* command, which shows information on the reservations that have been made by interface:

```
Router#show ip rsvp interface
interfac allocate i/f max  flow max per/255 UDP  IP   UDP_IP   UDP M/C
Et0      0M       1M       100K     0  /255 0    2    0        0
To0      50K      1M       100K     12 /255 0    1    0        0
```

This command shows that there are two interfaces that are currently supporting RSVP reservations, `Ethernet0` and `TokenRing0`. The *allocate* column shows the amount of bandwidth that has been allocated to active RSVP requests on each interface. In all of these fields, the letter K stands for Kbps and the M for Mbps. The *i/f max* column shows the total amount that can be allocated on each of these interfaces, while the *flow max* shows the maximum that can be requested by any one flow. These are the parameters from the *ip rsvp bandwidth* interface configuration command.

The remaining columns show information about the actual allocated streams. The *per/255* column shows the fraction of the total interface bandwidth that is used by each of these allocations. This is measured as a fraction of 255, as is common for expressing loads on Cisco interfaces. The *UDP* column shows the number of UDP-encapsulated sessions, *IP* counts the TCP-encapsulated sessions, *UDP_IP* shows the sessions that use both UDP and TCP. The *UDP M/C* column shows whether the interface is configured to allow UDP reservations.

You can look at individual reservations in detail with the following command:

```
Router#show ip rsvp installed
RSVP: Ethernet0 has no installed reservations
RSVP: TokenRing0
BPS    To              From            Protoc DPort  Sport  Weight Conversation
50K    192.168.5.5     192.168.1.10    TCP    888    999    4      520
Router#
```

This shows that the router is currently supporting a 50Kbps TCP session between the two IP addresses

that are shown, with the source and destination port numbers, 999 and 888, respectively. The *Weight* column shows the weighting factor, and *Conversation* shows the conversation (or flow) number used by WFQ for this queue. If you don't run WFQ on this interface, both of these values appear as 0.

There is considerable overlap between the information shown in the *show ip rsvp installed* command and the output with the *reservation* and *sender* keywords. However, there are some important additional pieces of information here:

```
Router#show ip rsvp reservation
To              From            Pro DPort Sport Next Hop      I/F   Fi Serv BPS Bytes
192.168.5.5     192.168.1.10    TCP 888   999   192.168.3.2    To0   FF LOAD 50K   50K
Router#show ip rsvp sender
To                From              Pro DPort Sport Prev Hop        I/F  BPS  Bytes
192.168.5.5       192.168.1.10      TCP 888   999   192.168.1.201   Et0   50K    50K

Router#
```

With the *reservation* keyword, you can see details about what type of reservation has been made. In this case, `FF` indicates that this is a *Fixed Filter* reservation, which means that it contains a single conversation between two end devices. However, RSVP also allows aggregation of flows. If this column says `SE`, which stands for *Shared Explicit Filter*, then it represents a shared reservation of unlimited scope. The other option is `WF` (Wildcard Filter) and indicates a shared reservation that can include only certain end devices or applications.

With the sender flag, you see the actual path information for the reservation. The Prev Hop and I/F columns here show the address and interface of the previous hop router. The *BPS* column shows the reserved bandwidth for this session in Kbps, and the *Bytes* column shows the maximum burst size in kilobytes.

The *show ip rsvp neighbor* command simply lists all of the IP addresses of active RSVP neighbors on all interfaces. This command is useful if you want to figure out what devices are making RSVP requests. As we mentioned earlier, since all RSVP requests are made hop-to-hop, it is quite likely that you will see a lot of routers in this list. However, on access routers, this command will help you to see whether the right end devices are making RSVP requests. If there are unauthorized devices in the list, you may want to consider using the *ip rsvp neighbor* interface configuration command to restrict which devices are allowed to make requests:

```
Router#show ip rsvp neighbor
Interfac Neighbor         Encapsulation
Et0      192.168.1.10     RSVP
Et0      192.168.1.201    RSVP
To0      192.168.3.2      RSVP
```

11.9.4 See Also

Recipe 11.6; Recipe 11.7; Recipe 11.8

Top

Recipe 11.10 Using Generic Traffic Shaping

11.10.1 Problem

You want to perform traffic shaping on an interface.

11.10.2 Solution

Generic traffic shaping works on an entire interface to limit the rate that it sends data. This first version restricts all outbound traffic to 500,000 bits per second:

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#interface FastEthernet0/0
Router(config-if)#traffic-shape rate 500000
Router(config-if)#end
Router#
```

You can also specify traffic shaping for packets that match a particular access list. This will buffer only the matching traffic, and leave all other traffic to use the default queueing mechanism for the interface:

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#access list 101 permit tcp any eq www any
Router(config)#access list 101 permit tcp any any eq www
Router(config)#access list 102 permit tcp any eq ftp any
Router(config)#access list 102 permit tcp any any eq ftp
Router(config)#interface FastEthernet0/0
Router(config-if)#traffic-shape group 101 100000
Router(config-if)#traffic-shape group 102 200000
Router(config-if)#end
Router#
```

There is also a newer class-based method for configuring traffic shaping on an interface using CBWFQ. We discuss this technique in Recipe 11.13.

11.10.3 Discussion

The first example shows how to configure an interface to restrict the total amount of outbound information. This is extremely useful when there is a device downstream that will not cope well with

hard bursts of traffic.

A common example is the method of delivering ATM WAN services through an Ethernet interface, frequently called *LAN Extension*. In this type of network, the Ethernet port on your router connects to the carrier's switch, which bridges one or more remote Ethernet segments together using an ATM network. The problem with this is that the Ethernet interface is able to send data much faster than the ATM network is configured to accept it. So you run the risk of dropping large numbers of packets within the ATM network. Since the carrier networks usually don't support customer Layer 3 QoS features, the entire ATM network acts just like a big FIFO queue with a tail drop problem. As we discuss in Appendix B, this is extremely inefficient.

This is why it can be extremely useful to restrict the total amount of traffic leaving an interface. It can also be useful to restrict only certain applications, as we demonstrated in the second example. However, we discuss more efficient class-based methods for controlling the total amount of traffic of a particular type in Recipe 11.7. This older group traffic-shaping method should only be used on routers that do not support CBWFQ.

11.10.4 See Also

Recipe 11.7; Recipe 11.12; Recipe 11.13

Top

Recipe 11.11 Using Frame-Relay Traffic Shaping

11.11.1 Problem

You want to separately control the amount of traffic sent along each of the PVCs in a Frame Relay network.

11.11.2 Solution

This first example shows how to configure Frame Relay traffic shaping using point-to-point frame relay subinterfaces:

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#interface HSSI0/0
Router(config-if)#encapsulation frame-relay
Router(config-if)#exit
Router(config)#interface HSSI0/0.1 point-to-point
Router(config-subif)#traffic-shape rate 150000
Router(config-subif)#frame-relay interface-dlci 31
Router(config-subif)#end
Router#
```

Most Frame Relay carrier networks are sufficiently overprovisioned, meaning that you can actually use much more capacity than your contractual Committed Information Rate (CIR). So you might want to apply traffic shaping only when you encounter Frame Relay congestion problems, and then only to reduce the data rate until the congestion goes away:

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#interface HSSI0/0
Router(config-if)#encapsulation frame-relay
Router(config-if)#exit
Router(config)#interface HSSI0/0.1 point-to-point
Router(config-subif)#traffic-shape adaptive 10000
Router(config-subif)#frame-relay interface-dlci 31
Router(config-subif)#end
Router#
```

11.11.3 Discussion

These examples are different from the one that we showed in Recipe 11.10. In this recipe we don't want to control the entire aggregate traffic flow, and we don't care about the traffic based on application. Here we want to ensure that every Frame Relay PVC using this interface is shaped separately so that they don't overrun the amount of bandwidth purchased from the WAN carrier. If you have 20 PVCs on an interface, it is fine to send the maximum per-PVC bandwidth to all of them simultaneously, but you will suffer from terrible performance problems if you try to send all of that bandwidth through a single PVC.

Usually you will purchase a particular amount of Frame Relay bandwidth, or CIR, from the WAN carrier for each PVC. So the first example shows how you can force the router to only send 150Kbps through the PVC with DLCI 31. It is important to remember that you can have different CIR values for different PVCs. You may need to have a different Frame Relay traffic shaping rate on every PVC.

The second example assumes that a lot of the time there will be very little congestion in the carrier's network, so you should be able to safely use some of the excess capacity. The Frame Relay protocol includes the ability to tell devices when there is congestion in the network. There are two types of congestion notifications, which are just noted as flags in the header portion of regular user frames. If a router receives a frame with the Forward Explicit Congestion Notification (FECN) flag set, it knows that the frame encountered congestion on its way from the remote device to the router. If the router receives a frame with the Backward Explicit Congestion Notification (BECN) flag set, a frame encountered congestion on its way from this router to the remote device. Please refer to Chapter 10 for a more detailed discussion of these Frame Relay protocol features.

The *traffic-shape adaptive* command tells the router that when it sees frames with a BECN flag, it should reduce the sending rate on this PVC. By default, this command will back off the sending rate all the way to zero. So, in the example, we have specified a minimum rate of 10,000bps, which would correspond to the CIR for this PVC:

```
Router(config-subif)#traffic-shape adaptive 10000
```

In general, this adaptive traffic shaping method is preferred over the static method, because it will give you significantly better network performance when the carrier's network is not congested. However, it is important to remember that the precise implementation of FECN and BECN markings is up to the carrier. Some carriers disable these features altogether, while others use them inconsistently. Since most customers ignore these markings, there is often very little reason to ensure that they are accurate.

You should check with your network vendor before implementing adaptive Frame Relay traffic shaping. We also recommend monitoring your FECN and BECN statistics for a reasonable period of time before implementing them, to verify that they are reliable.

11.11.4 See Also

Recipe 11.10; Recipe 11.12; Chapter 10

Top

Recipe 11.12 Using Committed Access Rate

11.12.1 Problem

You want to use Committed Access Rate (CAR) to control the flow of traffic through an interface.

11.12.2 Solution

CAR provides a useful method for policing the traffic rate through an interface. The main features of CAR are functionally similar to traffic shaping, but CAR also allows several extremely useful extensions. This first example shows the simplest application. We have configured CAR here to do basic rate limiting. The interface will transmit packets at an average rate of 500,000bps, allowing bursts of 4,500 bytes. If there is a burst of longer than 9,000 bytes, the router will drop the excess packets:

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#interface HSSI0/0
Router(config-if)#rate-limit output 500000 4500 9000  conform-action transmit exceed-
action drop
Router(config-if)#end
Router#
```

This next example defines three different traffic classifications using access lists, and separately limits the rates of these applications:

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#access list 101 permit tcp any eq www any
Router(config)#access list 101 permit tcp any any eq www
Router(config)#access list 102 permit tcp any eq ftp any
Router(config)#access list 102 permit tcp any any eq ftp
Router(config)#access list 102 permit tcp any eq ftp-data any
Router(config)#access list 102 permit tcp any any eq ftp-data
Router(config)#access list 103 permit ip any any
Router(config)#interface HSSI0/0
Router(config-if)#rate-limit output access-group 101  50000 4500 9000  conform-action
transmit exceed-action drop
Router(config-if)#rate-limit output access-group 102  50000 4500 9000  conform-action
transmit exceed-action drop
Router(config-if)#rate-limit output access-group 103  400000 4500 9000  conform-action
transmit exceed-action drop
Router(config-if)#end
Router#
```

CAR also includes a useful option to match DSCP in the rate-limit command without needing to resort

to an access group. In the following example, the DSCP values with the highest drop precedence values are rate-limited. Note that, unlike several other Cisco commands, here you must specify the decimal value of the DSCP field. Please refer to Table B-3 in Appendix B for a list of these values:

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#interface HSSI0/0
Router(config-if)#rate-limit output dscp 14  50000 4500 9000 conform-action  transmit
exceed-action drop
Router(config-if)#rate-limit output dscp 22  50000 4500 9000 conform-action transmit
exceed-action drop
Router(config-if)#rate-limit output dscp 30  50000 4500 9000 conform-action transmit
exceed-action drop
Router(config-if)#end
Router#
```

Finally, CAR also allows you to define a new kind of access list called a rate-limiting access list:

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#access list rate-limit 55 5
Router(config)#interface HSSI0/0
Router(config-if)#rate-limit output access-group rate-limit 55 50000 4500 9000
conform-action transmit exceed-action drop
Router(config-if)#end
Router#
```

11.12.3 Discussion

People are often confused about the difference between CAR and traffic shaping, because they appear to perform extremely similar functions. However, there is one very important difference. When a traffic shaping interface experiences a burst of data, it attempts to buffer the excess. But CAR just does whatever *exceed-action* you have specified:

```
Router(config-if)#rate-limit output 500000 4500 9000  conform-action transmit exceed-
action drop
```

In this example, the exceed-action is to simply drop the packet. Meanwhile, the *conform-action* in each example is to simply transmit the packet. Any traffic that falls below the configured rate is said to conform. CAR includes several other possibilities besides simply transmitting or dropping the packet:

drop

CAR drops the packet.

transmi t

CAR transmits the packet unchanged.

set-prec-transmit

CAR changes the IP Precedence of the packet, and then transmits it.

continue

CAR moves on to evaluate the next rate-limit command on this interface.

set-prec-continue

CAR changes the IP Precedence and then evaluates the next rate-limit command.

Cisco has added several additional options to IOS Versions 12.0(14)ST and higher:

set-dscp-continue

CAR changes the DSCP field and then evaluates the next rate-limit command.

set-dscp-transmit

CAR changes DSCP field and then transmits the packet.

set-qos-continue

CAR sets the qos-group and then evaluates next command.

set-qos-transmi t

CAR sets the qos-group and then transmits the packet.

There are two additional commands that you can use with MPLS to alter the MPLS experimental field:

set-mpls-exp-continue

This sets the experimental field and then continues.

set-mpls-exp-transmit

This option sets the experimental field and transmits the packet.

The various *continue* options allow you to string together a series of CAR commands on an interface to do more sophisticated things:

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#access list 101 permit tcp any eq www any
Router(config)#access list 101 permit tcp any any eq www
Router(config)#access list 103 permit ip any any
Router(config)#interface HSSI0/0
Router(config-if)#rate-limit output 50000 4500 4500  conform-action transmit exceed-
action continue
Router(config-if)#rate-limit output access-group 101  100000 4500 9000  conform-action
set-prec-transmit 3 exceed-action continue
Router(config-if)#rate-limit output access-group 103  100000 4500 9000  conform-action
set-prec-transmit 0 exceed-action drop
Router(config-if)#end
```

```
Router#
```

In this example, the interface will transmit all packets when the rate is 50,000bps or less. As soon as the traffic exceeds this rate, however, the router starts to bump up the IP Precedence of all HTTP traffic to a value of 3, and all other traffic goes down to a precedence of 0. It will continue to transmit all of these packets until the average rate exceeds 100,000bps. You can use this sort of technique to carefully tune how your network behaves in congestion situations.

You can also use CAR and the *exceed-action set-prec-transmit* command to lower the Precedence of high priority IP traffic when it exceeds its allocated portion of the bandwidth. Simply transmitting it with a lower Precedence represents a nice and useful intermediate step to dropping high priority packets outright. However, with real-time packets, it is better to drop than buffer or remark, because those options would introduce unwanted latency and jitter.

The other useful thing you can do with CAR is to rate-limit inbound traffic:

```
Router(config-if)#rate-limit input 50000 4500 4500 conform-action transmit exceed-
action drop
```

Of course, it's never completely ideal to allow a remote device to send too many packets across the network, only to drop them as they are received. But it is sometimes useful when your network acts as a service provider to other networks. For example, you might have downstream customers who have subscribed to a subrate service. This would include things like selling access through an Ethernet port, but restricting the customer to some lower rate such as 100Kbps.

Alternatively, you could use inbound rate-limit commands to ensure that your downstream customers are allowed to use your network for surfing the web, but only if the rate is kept below some threshold:

```
Router(config)#access list 101 permit tcp any eq www any
Router(config)#access list 101 permit tcp any any eq www
Router(config)#access list 103 permit ip any any
Router(config)#interface HSSI0/0
Router(config-if)#rate-limit input 50000 4500 4500 conform-action transmit exceed-
action continue
Router(config-if)#rate-limit input access-group 101  100000 4500 9000 conform-action
drop exceed-action continue
Router(config-if)#rate-limit input access-group 103  100000 4500 9000 conform-action
transmit exceed-action drop
Router(config-if)#end
Router#
```

You could even use CAR to simply rewrite the IP Precedence values of all packets received from a customer:

```
Router(config)#interface HSSI0/0
Router(config-if)#rate-limit input 100000 4500 9000 conform-action set-prec-transmit
0 exceed-action set-prec-transmit 0
Router(config-if)#end
```

```
Router#
```

This same technique is also helpful in combating Internet-based DOS attacks. For example, if your network is being inundated with PING flood or SYN ACK attacks, you might want to look specifically for these types of packets, and make sure that they are restricted to a low but reasonable rate. This way, the legitimate uses of these packets will not suffer, but you will reduce the service denial problem.

The last example in the solution section of this recipe needs a little bit of explanation because some of the properties can be confusing:

```
Router(config)#access list rate-limit 55 5
Router(config)#interface HSSI0/0
Router(config-if)#rate-limit output access-group rate-limit 55 50000 4500 9000
conform-action transmit exceed-action drop
```

The *access-list rate-limit* command allows you to create a new and special variety of access lists especially for use with CAR. There are three ranges of rate-limiting access list index numbers. You use access lists with values between 0 and 99 to match IP Precedence values. If the index number is between 100 and 199, it will match MAC addresses, and if it is between 200 and 299, it matches MPLS experimental field values.

In the example above, access list number 55 simply matches all packets with IP Precedence values of 5. You can also use a precedence bit mask to match several values. For example, to match Precedence values 0, 1, and 2, you could use a mask of`01100000` , which is `96` in decimal:

```
Router(config)#access list rate-limit 56 mask 96
```

The MPLS access lists work in a similar way, matching the value in the MPLS experimental field:

```
Router(config)#access list rate-limit 255 6
Router(config)#access list rate-limit 256 mask 42
```

The MAC address access lists work on standard Ethernet or Token Ring 48-bit MAC addresses:

```
Router(config)#access list rate-limit 155 0000.0c07.ac01
```

You have to be careful about how you use these rate-limiting access lists, because it's easy to get them confused with regular access lists. You can have a regular access list with the same number as a rate-limiting access list. The only difference is that you apply rate-limiting access lists with the rate-limit keyword on the rate-limit command as follows:

```
Router(config)#interface HSSI0/0
Router(config-if)#rate-limit output access-group rate-limit 55 50000 4500 9000
conform-action transmit exceed-action drop
```

Top

Recipe 11.13 Implementing Standards-BasedPer-Hop Behavior

11.13.1 Problem

You want to configure your router to follow the RFC-defined per-hop behaviors defined for different DSCP values.

11.13.2 Solution

This recipe constructs an approximate implementation of both expedited forwarding (EF) and assured forwarding (AF), while still ensuring that network control packets do not suffer from delays due to application traffic. With the QoS enhancements provided in IOS Version 12.1(5)T and higher, there is a straightforward way to accomplish this using a combination of WRED, CBWFQ and Low Latency Queueing (LLQ):

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#class-map EF
Router(config-cmap)#description Real-time application traffic
Router(config-cmap)#match ip precedence 5
Router(config-cmap)#exit
Router(config)#class-map AF1x
Router(config-cmap)#description Priority Class 1
Router(config-cmap)#match ip precedence 1
Router(config-cmap)#exit
Router(config)#class-map AF2x
Router(config-cmap)#description Priority Class 2
Router(config-cmap)#match ip precedence 2
Router(config-cmap)#exit
Router(config)#class-map AF3x
Router(config-cmap)#description Priority Class 3
Router(config-cmap)#match ip precedence 3
Router(config-cmap)#exit
Router(config)#class-map AF4x
Router(config-cmap)#description Priority Class 4
Router(config-cmap)#match ip precedence 4
Router(config-cmap)#exit
Router(config)#policy-map cbwfq_pq
Router(config-pmap)#class EF
Router(config-pmap-c)#priority 58 800
Router(config-pmap-c)#exit
Router(config-pmap)#class AF1x
```

```
Router(config-pmap-c)#bandwidth percent 15
Router(config-pmap-c)#random-detect dscp-based
Router(config-pmap-c)#exit
Router(config-pmap)#class AF2x
Router(config-pmap-c)#bandwidth percent 15
Router(config-pmap-c)#random-detect dscp-based
Router(config-pmap-c)#exit
Router(config-pmap)#class AF3x
Router(config-pmap-c)#bandwidth percent 15
Router(config-pmap-c)#random-detect dscp-based
Router(config-pmap-c)#exit
Router(config-pmap)#class AF4x
Router(config-pmap-c)#bandwidth percent 15
Router(config-pmap-c)#random-detect dscp-based
Router(config-pmap-c)#exit
Router(config-pmap)#class class-default
Router(config-pmap-c)#fair-queue 512
Router(config-pmap-c)#queue-limit 96
Router(config-pmap-c)#exit
Router(config-pmap)#exit
Router(config)#interface HSSI0/1
Router(config-if)#service-policy output cbwfqpolicy
Router(config-if)#end
Router#
```

If you are running older IOS versions, you can use Custom Queueing to create different levels of forwarding precedence, but you can't combine them with WRED to enforce the standard drop precedence rules.

11.13.3 Discussion

We have repeatedly said throughout this chapter that Priority Queues are dangerous because they allow high priority traffic to starve all lower priority queues. However, strict prioritization does give excellent real-time behavior for the highest priority traffic because it ensures minimal queueing latency. This recipe uses Cisco's low LLQ, which avoids most of the problems of pure Priority Queueing.

This example creates an approximation of the differentiated services model defined in RFCs 2597 and 2598. All real-time and network control traffic uses LLQ to ensure that it is always delivered with minimal delay. All other IP traffic falls into one of the assured forwarding classes shown in Appendix B, with the exception of packets that do not have a DSCP tag value. Untagged traffic, including non-IP traffic, will use the default forwarding behavior.

Each column in Table B.3 represents a precedence class, which we have called AF1x, AF2x, AF3x, and AF4x. For each of these classes, we have reserved a share of the bandwidth and configured DSCP-based WRED:

```
Router(config-pmap)#class AF4x
```

```
Router(config-pmap-c)#bandwidth percent 15
Router(config-pmap-c)#random-detect dscp-based
Router(config-pmap-c)#exit
```

We showed how to modify the default WRED thresholds and drop probabilities in the discussion section of Recipe 11.8.

The example also defines a class called EF that matches all packets with an IP Precedence value of 5, which has a binary representation of `101`. Note that, technically, the EF DSCP value looks like `101110` in binary. So, the example allows packets to join this queue if only the first 3 bits of the DSCP are correct. This is for backward compatibility and to ensure that we don't leave out any high priority traffic. However, if you wanted to create a queue for a real EF DSCP value and a separate queue for packets with IP Precedence 5, you could do so like this:

```
Router(config)#class-map EF
Router(config-cmap)#description Real-time application traffic
Router(config-cmap)#match ip dscp ef
Router(config-cmap)#exit
Router(config)#class-map Prec5
Router(config-cmap)#description Critical application traffic
Router(config-cmap)#match ip precedence 5
Router(config-cmap)#exit
Router(config)#policy-map cbwfq_pq
Router(config-pmap)#class EF
Router(config-pmap-c)#priority 58 800
Router(config-pmap-c)#exit
Router(config-pmap)#class Prec5
Router(config-pmap-c)#bandwidth percent 15
Router(config-pmap-c)#exit
```

Note that you must define the classes in the policy map in this order because the router matches packets sequentially. If you specified the EF class map after the `Prec5` map, you would find that all of your EF traffic would wind up in the other queue, which is not what you want. Note also that, as we discussed in Recipe 11.7, you have to be careful of the total reserved bandwidth in CBWFQ. Simply adding these lines to the recipe example would give a total of 75% allocated bandwidth. This is the default maximum value. If you want to exceed this value, Recipe 11.7 shows how to modify the maximum.

We have already discussed most of the commands shown in the class definitions in other recipes. However, the EF queue contains a special command:

```
Router(config-pmap-c)#priority 58 800
```

This defines a strict priority queue for this class with a sustained throughput of 58Kbps. This is based on the assumption that the EF application uses a standard stream of 56Kbps, and we have added a small amount on top of this to allow for Layer 2 overhead. The last argument in the *priority* command is a burst length in bytes. This allows the application to temporarily exceed the defined sustain rate, just long enough to send this many bytes. In this case we're assuming that the real-time application uses small

packets, so allowing it to send a burst of 800 bytes when it has already reached the configured sustain rate of 58Kbps should be more than sufficient.

Note that this does not imply that there is strict policing on this queue. If you also want to enforce a maximum rate of 65Kbps, for example, on this queue, you could also include a *police* statement, as follows:

```
Router(config-pmap)#class EF
Router(config-pmap-c)#priority 58 800
Router(config-pmap-c)#police 65000 1600
Router(config-pmap-c)#exit
```

In this command we have also been slightly more generous with the burst size, extending it to 1600 bytes. Enforcing an upper limit like this is a good idea on priority queues because it helps to prevent the highest priority traffic from starving the other queues. However, it is not necessary to enforce the upper limit this way just to avoid starving the lower queues. This is because the *priority* command stops giving strict priority to this queue when the bandwidth rises above the specified limit.

It should be clear from the ability to allocate both minimum and maximum bandwidth with the CBWFQ *priority* and *police* commands, LLQ is a much more sophisticated and flexible type of queueing than the simple Priority Queueing discussed in Recipe 11.3. So if your router is capable of supporting CBWFQ, we recommend using LLQ for any situation where you want Priority Queueing. Cisco introduced the LLQ feature in IOS level 12.0(6)T.

11.13.4 See Also

Recipe 11.3; Recipe 11.5; Recipe 11.7

Top

Recipe 11.14 Viewing Queue Parameters

11.14.1 Problem

You want to see how queueing is configured on an interface.

11.14.2 Solution

Cisco provides several useful commands for looking at an interface's queueing configuration and performance. The first of these is the *show queue* command:

```
Router#show queue FastEthernet0/0
  Input queue: 0/75/105/0 (size/max/drops/flushes); Total output drops: 0
  Queueing strategy: weighted fair
  Output queue: 0/1000/96/0 (size/max total/threshold/drops)
     Conversations  0/1/128 (active/max active/max total)
     Reserved Conversations 0/0 (allocated/max allocated)
     Available Bandwidth 75000 kilobits/sec


Router#
```

Use the *show queueing* command to look the router's queueing configuration in general:

```
Router#show queueing
Current fair queue configuration:

  Interface          Discard    Dynamic  Reserved  Link    Priority
                     threshold  queues   queues    queues  queues
  FastEthernet0/0    96         128      258       8       1
  Serial0/0          64         256      37        8       1
  Serial0/1          96         128      256       8       1

Current DLCI priority queue configuration:
Current priority queue configuration:

List   Queue  Args
1      high   protocol ip          tcp port 198
1      high   protocol pppoe-sessi
2      high   protocol ip          udp port 199
3      low    default
3      high   protocol ip          list 101
Current custom queue configuration:
```

```
Current random-detect configuration:
Router#
```

11.14.3 Discussion

The *show queue* and *show queueing* commands augment the *show interface* output, which also shows important queueing information:

```
Router#show interface FastEthernet0/0
FastEthernet0/0 is up, line protocol is up
  Hardware is AmdFE, address is 0001.9670.b780 (bia 0001.9670.b780)
  MTU 1500 bytes, BW 100000 Kbit, DLY 100 usec,
     reliability 255/255, txload 1/255, rxload 1/255
  Encapsulation ARPA, loopback not set
  Keepalive set (10 sec)
   Full-duplex, 100Mb/s, 100BaseTX/FX
  ARP type: ARPA, ARP Timeout 04:00:00
  Last input 00:00:00, output 00:00:00, output hang never
  Last clearing of "show interface" counters never
  Input queue: 0/75/105/0 (size/max/drops/flushes); Total output drops: 0
  Queueing strategy: weighted fair
  Output queue: 0/1000/96/0 (size/max total/threshold/drops)
     Conversations  0/1/128 (active/max active/max total)
     Reserved Conversations 0/0 (allocated/max allocated)
     Available Bandwidth 75000 kilobits/sec
  5 minute input rate 1000 bits/sec, 2 packets/sec
  5 minute output rate 2000 bits/sec, 2 packets/sec
     2495069 packets input, 181306312 bytes
     Received 2333309 broadcasts, 0 runts, 0 giants, 0 throttles
     0 input errors, 0 CRC, 0 frame, 0 overrun, 0 ignored
     0 watchdog
     0 input packets with dribble condition detected
     1927544 packets output, 197958017 bytes, 0 underruns
     0 output errors, 0 collisions, 21 interface resets
     0 babbles, 0 late collision, 0 deferred
     0 lost carrier, 0 no carrier
     0 output buffer failures, 0 output buffers swapped out
Router#
```

The *show queue* command is a good starting point when looking at queueing issues. It tells you what queueing algorithm is used, as well as information about any drops:

```
Router#show queue FastEthernet0/0
  Input queue: 0/75/105/0 (size/max/drops/flushes); Total output drops: 0
  Queueing strategy: weighted fair
  Output queue: 0/1000/96/0 (size/max total/threshold/drops)
     Conversations  0/1/128 (active/max active/max total)
     Reserved Conversations 0/0 (allocated/max allocated)
     Available Bandwidth 75000 kilobits/sec
```

In this case, you can see that the interface uses WFQ. This can be slightly deceptive because we actually configured this interface for CBWFQ. The `Reserved Connections` line indicates no RSVP reservation queues have been allocated for this interface. So, if you tried to use RSVP on this interface, it would not work right now.

The *show queue* command gives no output at all when you use Custom Queueing or Priority Queueing on an interface.

The first section of output from the *show queueing* command gives some useful summary information on fair queueing parameters:

```
Router#show queueing
Current fair queue configuration:

  Interface             Discard    Dynamic  Reserved  Link     Priority
                        threshold  queues   queues    queues   queues
  FastEthernet0/0       96         128      258       8        1
  Serial0/0             64         256      37        8        1
  Serial0/1             96         128      256       8        1
```

In this case you can immediately see and compare the queue sizes between different interfaces.

Top

Chapter 12. Tunnels and VPNs

Top

Introduction

A tunnel is essentially just a method for encapsulating one protocol in another. There are many reasons for doing this. In Chapter 15 we will discuss DLSw, which is commonly used to transmit SNA traffic through an IP network. The SNA protocol is not routable, so the tunnel allows you to send this traffic through a scalable routed network.

You can also use tunnels to transmit protocols that are routable, but not fully supported by the network. For example, some organizations find that they need to be able to send IPX through their networks to support legacy applications. But few network engineers are willing to invest the extra time or money required to build native IPX support into their routing core. So this is an ideal situation for using tunnels.

And we often see tunnels for carrying IP traffic through an IP network. The classic example of this is a Virtual Private Network (VPN) that connects two private networks through a public network such as the Internet. But there are other places where it can be useful to tunnel IP in IP.

One of the most common reasons for tunneling IP in IP is to get around architectural problems with dynamic routing protocols. For example, in Chapter 8 we discussed OSPF virtual links. These are effectively just tunnels that let you put routers in different OSPF areas than their physical connections allow.

Another example appears when you need to extend a routing protocol through regions of the network that don't support this protocol. Some WAN carriers provide IP connectivity between customer locations, similar to a public Internet. But the carrier network can't always support the customer's routing protocol and it is often not desirable to mix the carrier and customer routing tables.

Tunnels are extremely useful in lab or test environments where they allow you to emulate more complex network topologies. Further, in lab environments it is sometimes necessary to tunnel test data through a production network to ensure that the testing cannot interfere with the functioning of the production network. We expect to see a lot of tunneling during the migration phases of any future large scale conversions to IPv6.

Most of the examples in this chapter will look at Generic Routing Encapsulation (GRE) tunnels, sometimes with encryption support using IPSec. GRE is an open standard documented in RFCs 1701 and 1702, and updated in RFC 2784. These documents actually describes GRE Version 0, which is the standard version of GRE. There is also a GRE Version 1, which is more commonly called PPTP (Point-to-Point Tunneling Protocol), and is described in RFC 2637. The key different between GRE and PPTP is that PPTP includes a PPP intermediate layer, while GRE directly supports Layer 3 protocols such as IP and IPX. This chapter does not have the space to cover PPTP or its cousins L2TP (Layer 2 Tunneling

Protocol) and L2F (Layer 2 Forwarding). These protocols are commonly used in situations where mobile users need to make VPN connections through the public Internet to an enterprise IP network. There are simply too many different variations to adequately cover even the most common configurations.

GRE doesn't use TCP or UDP. Instead, this protocol works directly with the IP layer, using IP Protocol number 47. It includes its own features for verifying delivery and integrity. The GRE packet's payload includes a complete Layer 3 packet with its payload and headers intact. The routers that terminate the tunnel take packets and wrap them in a new IP packet with a GRE header. They forward this GRE packet through the IP network to the router that supports the other end of the tunnel. The receiving router then simply unwraps the encapsulated packet and sends it on its way. To the encapsulated packet, this entire process has taken a single routing hop, even though the GRE packet may have traversed many routers to reach its destination.

There are other common tunnel protocols, such as IP-in-IP, which uses IP protocol number 4. This protocol is an open standard that is documented in RFC 2003. In general we prefer GRE to IP-in-IP because it offers considerably greater flexibility, particularly on Cisco routers.

Tunnels can have packet fragmentation issues. The problem is simply that when you put a second IP header on an existing packet, you get a bigger packet. If the original packet is already at or close to the Maximum Transmission Unit (MTU) packet size that the network can support, putting this packet in a tunnel forces the router to fragment it. Most of the time this is not a problem, but some applications do not cope well with packet fragmentation.

Normally, applications that can't accept packet fragmentation will set the Don't Fragment (DF) bit in the IP header. The router must drop oversized packets that it cannot fragment, but it sends an ICMP message back to the end device to tell it to use a smaller packet size.

The net result is that when you use tunnels, you reduce the effective MTU of your network. This doesn't necessarily cause problems, but it is important to be aware of the consequences.

Internet Protocol Security (IPSEC) is a suite of security related protocols and algorithms documented in RFCs 2401 through 2412 and RFC 2451. This is far more information than we can even summarize in a book like this, so we mention only some of the most immediately relevant points. The RFCs or books such as IPSec: Securing VPNs (RSA Press) are good resources for more information.

The IPSec framework provides features for authenticating and encrypting traffic as well as for securely exchanging encryption and authentication keys. It is designed to work with both IPv4 and IPv6, and can accommodate a variety of different basic encryption, authentication, and key exchange algorithms. This algorithmic independence is one of the essential design criteria of IPSec. It allows you to transparently substitute a new encryption algorithm, for example, if somebody discovers a critical flaw in the old one, or if a new algorithm is more efficient.

IPSec provides security only at the IP layer. This allows it to protect applications and data operating at higher layers of the protocol stack. This is important because it means that you can use IPSec in

conjunction with other insecure protocols or applications and, if done properly, achieve a good level of overall security. Also, because IPSec works at the IP layer, you can readily use it with any of the higher layer IP-based protocols such as TCP, UDP, ICMP, multicast, and so forth.

Unfortunately, one of the most confusing things about IPSec is the proliferation of different protocols and algorithms that handle different parts of the key management, authentication, and encryption processes. Therefore, we will briefly explain some of the more common terms and concepts.

Internet Security Association Key Management Protocol (ISAKMP) is essentially a framework for key exchange, a generic set of procedures and packet formats that allow devices to reliably and securely pass encryption and authentication keys to one another. It includes such concepts as the key security association, which defines not only the keys themselves but important parameters such as the specific algorithms to be used and the length of time that this key is valid for. This information is all negotiated by the IPSec end devices when they first establish a session, and periodically updated if the session remains active for a longer period of time.

Internet Key Exchange (IKE) is a specific protocol for securely exchanging keys using the ISAKMP framework. It uses the OAKLEY key determination protocol, which is defined in RFC 2412. OAKLEY distributes keys of arbitrary types for arbitrary algorithms to use. One of the methods that it can use is the Diffie-Hellman (DH) key exchange model.

DH is a mathematical algorithm that uses properties of large prime numbers to allow users to exchange key information in encrypted form. Both devices authenticating a session can calculate a common key based on the encrypted information that they exchange. There are two issues with this algorithm.

The first is that it can be broken by a "man in the middle" attack. This essentially involves somebody intercepting the exchanged key information and rewriting it to create a new valid key with each of the end devices. The various key management protocols get around this problem by using a separate authentication system to validate the exchanged information.

The second problem is that even with authentication, if the prime numbers aren't large enough, it is possible to mathematically deduce the key. To resolve this problem, Cisco routers offer several different DH Groups. Group 1 uses 768-bit values to define the prime numbers, Group 2 uses 1024-bit primes. And, in IOS level 12.1T, Cisco introduced support for Group 5 DH, which uses a 1536-bit value for its prime numbers. With current computing power, if somebody really wants your data, 768- bit values are not very secure. So we recommend using Group 2 or higher.

OAKLEY also supports the Perfect Forward Secrecy (PFS) system. PFS is a system that ensures that even if somebody is able to break one of the keys, this will tell them nothing about any other keys. This is because the keys are not derived from one another. Many of the Cisco commands related to key exchange include a *pfs* keyword that you can enable, although you need to ensure that the same options are enabled on both peers.

One of the most effective ways of managing large numbers of keys is to implement a Public Key

Infrastructure (PKI), which is a paradigm that uses digital certificates to verify the validity of public encryption and authentication keys. They generally use a Certification Authority (CA), which is a trusted server that knows the public encryption keys for a large number of devices.

IPSec uses two important security protocols, the Authenticating Header (AH), and the Encapsulating Security Payload (ESP). These do pretty much what their names suggest. AH includes a cryptographic authentication scheme in the header of the IP packet, which allows you to ensure that the data has not been tampered with in any way, and that it really does come from the right source device. ESP, on the other hand, encrypts the packet's payload for privacy. We recommend that if you are using IPSec, you should use both AH and ESP together. Authentication and encryption clearly serve entirely different but complementary functions, but we believe it is rare to have data that is important enough to warrant implementing either authentication or encryption but not both.

One of the main authentication methods for IPSec makes use of a cryptographic hash function. Hash functions are actually more common than you might think. The simple Cyclic Redundancy Checksum (CRC) field in a packet is essentially just a hash function. The general definition of a hash function is an algorithm that takes a message of arbitrary length and produces an output of fixed length. This output is often called a message digest.

To be useful for authentication, this hash function must make it extremely difficult to generate a two distinct messages that have the same message digest. There are several of these hash functions in existence. The most popular for use with IPSec are Message Digest Version 5 (MD5) and Secure Hash Algorithm (SHA). We have already discussed MD5 in another setting, when we talked about how Cisco routers store passwords internally in Chapter 3. Cryptographic hash functions make excellent password crypts, because the result is always the same length and almost impossible to reverse. If the algorithm is strong, the only way to decrypt the original password is to encrypt a series of systematic guesses and see if any of them match the unknown encrypted string.

The National Institute of Standards and Technology (NIST) developed the SHA as an improvement over MD5. It is generally believed that SHA is somewhat more secure than MD5, although it is a little bit more CPU intensive.

IPSec uses these hash functions to create Hashed Message Authentication Codes (HMAC). The HMAC is effectively an irreversible cryptographic hash function of an original message that has been combined in a nontrivial way with a password. So you need to not only break the hash algorithm, but also the password to reconstruct the original message.

For the actual data encryption, IPSec again offers several different options. Cisco routers only implement 56-bit and 168-bit Data Encryption Standard (DES) encryption. The 56-bit version of DES is the default, while the 168-bit version is often called Triple DES, and has export restrictions outside of North America.

People have developed several other encryption algorithms for use with IPSec. One of the most popular

is called Blowfish. This is an unpatented and freely distributable encryption algorithm that is faster than standard DES, and believed to be more secure as well. Other encryption algorithms include International Data Encryption Algorithm (IDEA), CAST-256, and Skipjack. However, Cisco implements only DES and Triple DES.

IPSec has two main modes of operation: tunnel mode and transport mode. In this chapter we will discuss examples of both. Tunnel mode essentially means that IPSec is responsible for operating its own tunnel. IPSec tunnels are modeled on the IP-in-IP tunnel protocol, which we mentioned earlier. As a result, any IPSec exchanges that use transport mode must be purely between the two end devices, while tunnel mode can support routing from devices that are further downstream. In Recipe 12.3, we will show an example where transport mode is used to encrypt traffic in a GRE tunnel. In this case, the GRE traffic always begins and ends on the routers themselves, although the payload of the GRE packets may contain IP packets routed from other downstream devices. Recipe 12.6, on the other hand, shows an example of tunnel mode. In this case, a remote workstation initiates the IPSec connection to the router. But the packets that this workstation sends are destined for end devices on the other side of the router. So tunnel mode is appropriate here.

By default, Cisco routers will use IPSec in tunnel mode. This is because IPSec needs a well-defined starting and ending point for the encryption. So, with transport mode, the source and destination IP addresses must be fixed somehow. This effectively means that transport mode needs to operate inside of another tunnel protocol such as GRE if it is to carry user traffic between routers.

Top

Recipe 12.1 Creating a Tunnel

12.1.1 Problem

You want to tunnel IP traffic through your network.

12.1.2 Solution

The basic GRE tunnel configuration is simply a matter of defining the source and destination addresses or interfaces on both devices. On the first router you need to create the tunnel interface, and define its source and destination:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#interface Tunnel1
Router1(config-if)#ip address 192.168.35.6 255.255.255.252
Router1(config-if)#tunnel source 172.25.1.5
Router1(config-if)#tunnel destination 172.25.1.7
Router1(config-if)#end
Router1#
```

Then, on the other router, you must create a tunnel interface with a matching source and destination:

```
Router5#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router5(config)#interface Tunnel3
Router5(config-if)#ip address 192.168.35.5 255.255.255.252
Router5(config-if)#tunnel source 172.25.1.7
Router5(config-if)#tunnel destination 172.25.1.5
Router5(config-if)#end
Router5#
```

12.1.3 Discussion

Creating a basic tunnel is very simple, you just need to define a source and destination on each of two routers. As with any other virtual interface such as subinterfaces or loopback interfaces, a tunnel has an additional memory requirement on the router. However, the CPU overhead is not as bad as you might initially think. This is because GRE tunnels do work well with Cisco Express Forwarding (CEF). So the main scaling issue in creating tunnels on routers is the memory required to support them.

The only tricky part of configuring a tunnel is making sure that the source of the tunnel on one router matches the destination on the other. In this case Router1 uses a source IP address of `172.25.1.5`, which happens to be its Ethernet port. If you look at the *tunnel destination* command on the other

router, you will see that it matches. Similarly, the destination on the first router is `172.25.1.7` , and the source is `172.25.1.5` .

You could also use an alternative syntax, specifying the interface name rather than the IP address:

```
Router5(config)#interface Tunnel3
Router5(config-if)#tunnel source Ethernet0
```

This points the tunnel source to the primary IP address on a particular interface on this router. It is crucial that this IP address must match the destination address configured on the other router.

If you then look at the new tunnel interface, you will see that it is up:

```
Router1#show interfaces tunnel1
Tunnel1 is up, line protocol is up
  Hardware is Tunnel
  Internet address is 192.168.35.6/30
  MTU 1514 bytes, BW 9 Kbit, DLY 500000 usec,
     reliability 255/255, txload 1/255, rxload 1/255
  Encapsulation TUNNEL, loopback not set
  Keepalive not set
  Tunnel source 172.25.1.5 (FastEthernet0), destination 172.25.1.7
  Tunnel protocol/transport GRE/IP, key disabled, sequencing disabled
  Checksumming of packets disabled,  fast tunneling enabled
  Last input 00:11:08, output 00:00:08, output hang never
  Last clearing of "show interface" counters never
  Input queue: 0/75/0/0 (size/max/drops/flushes); Total output drops: 0
  Queueing strategy: fifo
  Output queue: 0/0 (size/max)
  5 minute input rate 0 bits/sec, 0 packets/sec
  5 minute output rate 0 bits/sec, 0 packets/sec
     5 packets input, 740 bytes, 0 no buffer
     Received 0 broadcasts, 0 runts, 0 giants, 0 throttles
     0 input errors, 0 CRC, 0 frame, 0 overrun, 0 ignored, 0 abort
     73 packets output, 6604 bytes, 0 underruns
     0 output errors, 0 collisions, 0 interface resets
     0 output buffer failures, 0 output buffers swapped out
Router1#
```

This is deceptive, though. Even if we remove the tunnel configuration from the other router, this interface will still appear to be up. Indeed, this tunnel interface will appear to be up even if you turn off the power on the far end router. In IOS Version 12.2(8)T, Cisco introduced a new *keepalive* option for GRE tunnels that overcomes this limitation. When you configure a tunnel with this new feature, the interface will go down if there are any connection problems:

```
Router1(config)#interface Tunnel1
Router1(config-if)#keepalive
```

By default, this *keepalive* command sends a packet through the tunnel to check its status once every 10 seconds. If there is no response to three successive polls, the router declares the tunnel interface to be

down. So, this will change the tunnel's status about 30 seconds after a failure.

You can adjust both the time interval and the number of retries. For example, to send a keepalive packet every five seconds, but to keep the default three retry limit, you could use the following command:

```
Router1(config)#interface Tunnel1
Router1(config-if)#keepalive 5
```

If you want to change the number of retries, you can specify the new value after the time interval. The following example will send a keepalive packet every three seconds, and will declare the tunnel down if it doesn't hear a response back to two successive keepalive tests:

```
Router1(config)#interface Tunnel1
Router1(config-if)#keepalive 3 2
```

If you are concerned about the integrity of tunneled data, you can enable checksums on a GRE tunnel:

```
Router1(config)#interface Tunnel1
Router1(config-if)#tunnel checksum
```

When you turn on checksums, the router will verify the checksum of every GRE packet it receives and drop any packets that don't match. A similar feature checks to see if packets are received in the correct order:

```
Router1(config)#interface Tunnel1
Router1(config-if)#tunnel sequence-datagrams
```

When you enable the *sequence-datagrams* option, the router will drop any packets that it receives out of their correct order. These two options can be useful in networks that have a tendency to damage packets, or where there are multiple paths between the tunnel routers. Remember that GRE doesn't use TCP, so these features can help to improve the reliability of a tunnel connection. However, even when you enable these features, the routers will not resend dropped packets (as TCP does).

We do suggest using some caution when you enable either checksums or sequencing on a GRE tunnel, because these features do not work with CEF. So, as soon as you enable either of them, the router will have to resort to process switching, which could drive up your CPU utilization.

The tunnel in the recipe didn't specify a particular tunnel protocol, so the routers will use the default GRE protocol. If you prefer to use a different tunnel protocol, you can change it using the tunnel mode command as follows:

```
Router1(config)#interface Tunnel1
Router1(config-if)#tunnel mode ipip
```

Here we have opted to use the IP-in-IP tunnel protocol that we discussed in the introduction to this chapter. There are several other options for tunnel protocols, which we list in Table 12-1.

Table 12-1. Available tunnel modes

Command	Description
Router1(config-if)#**`tunnel mode aurp`**	AppleTalk "TunnelTalk" encapsulation
Router1(config-if)#**`tunnel mode cayman`**	Cayman AppleTalk tunnel encapsulation
Router1(config-if)#**`tunnel mode dvmrp`**	DVMRP multicast tunnel
Router1(config-if)#**`tunnel mode eon`**	Allows tunneling of CLNP OSI-based protocols through IP networks
Router1(config-if)#**`tunnel mode gre ip`**	GRE encapsulation, the default
Router1(config-if)#**`tunnel mode gre ip multipoint`**	GRE encapsulation; multipoint option automatically creates a mesh of tunnels among the participating routers
Router1(config-if)#**`tunnel mode ipip`**	IP in IP encapsulation
Router1(config-if)#**`tunnel mode iptalk`**	AppleTalk IPTalk encapsulation
Router1(config-if)#**`tunnel mode nos`**	A version of IP in IP that supports the KA9Q protocol

In the recipe example, the two routers shared an Ethernet segment, so the routing was trivial. But, in practice, routing between the tunnel end points is often the most difficult thing to get right. The problem, which we will discuss more in Recipe 12.3, is that the tunnel itself is only one routing hop for packets that travel through it, although it might be several physical hops. As a result, the routing protocol will often decide that the best way to get to the tunnel's destination IP address is through the tunnel itself. This is called *recursive routing* , and it makes the tunnel useless. So, when a router notices that it is routing GRE packets for a tunnel destination address through the same tunnel, it will automatically disable the tunnel with the following error message:

```
Jan 16 12:05:04 EST: %TUN-5-RECURDOWN: Tunnel1 temporarily disabled due to recursive
routing
Jan 16 12:05:05 EST: %LINEPROTO-5-UPDOWN: Line protocol on Interface Tunnel1, changed
state to down
```

Cisco has attempted to reduce this problem by making the default bandwidth for all tunnel interfaces 9Kbps. For most routing protocols, this means that you have to traverse several hops before the tunnel looks like a better path. But some protocols, most notably RIP, don't look at interface bandwidths. It is

important to bear in mind that, no matter what protocol you use, at some point a single hop of 9Kbps is going to look better than a large number of higher bandwidth hops.

The only way to avoid this problem is to ensure that there is always a good route to the tunnel destination that doesn't use the tunnel itself. This is most easily accomplished using static routes, although we will discuss other techniques in Recipe 12.3.

12.1.4 See Also

Recipe 12.3

Top

Recipe 12.2 Tunneling Foreign Protocols in IP

12.2.1 Problem

You want to tunnel a foreign protocol (such as IPX traffic) through your IP network.

12.2.2 Solution

One of the most important applications of tunnels is for passing foreign protocols through a network that only supports IP. A typical example of this would be IPX, although the configuration is similar for other protocols such as AppleTalk:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#ipx routing AAAA.BBBB.0001
Router1(config)#interface Tunnel1
Router1(config-if)#ipx network AAA
Router1(config-if)#tunnel source 172.25.1.5
Router1(config-if)#tunnel destination 172.25.1.7
Router1(config-if)#end
Router1#
```

Then, on the other router, you must create a tunnel interface with a matching source and destination as well as matching IPX network number:

```
Router5#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router2(config)#ipx routing AAAA.BBBB.0002
Router5(config)#interface Tunnel3
Router5(config-if)#ipx network AAA
Router5(config-if)#tunnel source 172.25.1.7
Router5(config-if)#tunnel destination 172.25.1.5
Router5(config-if)#end
Router5#
```

12.2.3 Discussion

This recipe is nearly identical to Recipe 12.1, but instead of tunneling IP traffic through an IP network, we use the same kind of tunnel to pass IPX traffic through the same network. Note that of all the supported tunnel modes mentioned in Table 12-1, only the default GRE will transport IPX (although there are several AppleTalk tunnel modes).

This book does not cover IPX, so we won't go into any detail on the IPX-specific commands here. This is merely intended as an example of how to use GRE tunnels for foreign protocols. For more information on IPX, please refer to Designing Large-Scale LANs (O'Reilly).

To enable IPX on both of these routers, you have to first make sure that you are running an IOS release that supports IPX. The various "Desktop" IOS versions support this protocol, as do the "Enterprise" versions. Please consult the Cisco IOS feature matrixes for more details. Assuming, then, that your router supports IPX, you can enable it with the *ipx routing* command, as shown. Naturally, you need to enable IPX routing on both routers. Then, the only other important detail is to configure both ends of the GRE tunnel with matching IPX network numbers as we have done in the example.

It's important to note that you can configure a GRE tunnel to support more than one protocol by simply specifying appropriate network numbers for each protocol using the tunnel. We could, for example, add IP to this IPX tunnel by simply configuring an IP address on both ends, as we did in Recipe 12.1. The tunnel will then support both protocols simultaneously:

```
Router1(config)#interface Tunnel1
Router1(config-if)#ip address 192.168.35.6 255.255.255.252
Router1(config-if)#ipx network AAA
Router1(config-if)#tunnel source 172.25.1.5
Router1(config-if)#tunnel destination 172.25.1.7
Router1(config-if)#end
Router1#
```

12.2.4 See Also

Recipe 12.1; Designing Large-Scale LANs, by Kevin Dooley (O'Reilly)

Top

Recipe 12.3 Tunneling with Dynamic Routing Protocols

12.3.1 Problem

You need to pass a dynamic routing protocol through your tunnels.

12.3.2 Solution

Dynamic routing and tunnels can be a dangerous combination. It is critical to ensure that the routers never get confused and think that the best path to the tunnel destination is through the tunnel itself. We offer three different ways of resolving this problem.

The first is to use static routes for the tunnel destination address:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#interface Tunnel1
Router1(config-if)#ip address 192.168.35.6 255.255.255.252
Router1(config-if)#tunnel source 172.25.1.5
Router1(config-if)#tunnel destination 172.22.1.2
Router1(config-if)#exit
Router1(config)#ip route 172.22.1.2 255.255.255.255 172.25.1.1
Router1(config)#router eigrp 55
Router1(config-router)#network 192.168.35.0
Router1(config-router)#end
Router1#
```

The second method simply excludes the tunnel's IP address range from the routing protocol. You can then run a different routing protocol for the addresses that you want to pass through the tunnel:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#interface Tunnel1
Router1(config-if)#ip address 192.168.35.6 255.255.255.252
Router1(config-if)#tunnel source 172.25.1.5
Router1(config-if)#tunnel destination 172.22.1.2
Router1(config-if)#exit
Router1(config)#router eigrp 55
Router1(config-router)#network 172.22.0.0
Router1(config-router)#network 172.25.0.0
Router1(config-router)#end
Router1(config)#router rip
```

```
Router1(config-router)#network 192.168.35.0
Router1(config-router)#end
Router1#
```

The third solution is to filter the routes of the supporting network to prevent them from passing through the tunnel:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#interface Tunnel1
Router1(config-if)#ip address 192.168.35.6 255.255.255.252
Router1(config-if)#tunnel source 172.25.1.5
Router1(config-if)#tunnel destination 172.22.1.2
Router1(config-if)#exit
Router11(config)#ip prefix-list TUNNELROUTES seq 10 permit  192.168.0.0/16 ge 17
Router1(config)#router eigrp 55
Router1(config-router)#network 172.22.0.0
Router1(config-router)#network 172.25.0.0
Router1(config-router)#network 192.168.35.0
Router1(config-router)#distribute-list prefix TUNNELROUTES out Tunnel1
Router1(config-router)#end
Router1#
```

12.3.3 Discussion

As we mentioned in Recipe 12.1, you have to be careful when using a dynamic routing protocol anywhere near a GRE tunnel to avoid the dreaded *recursive routing* error message, which brings down the tunnel. This happens because the routers need to have a good path through the network to carry the tunnel to its destination. In addition, this path cannot go through the tunnel itself. But the problem is that, because the tunnel forms a virtual connection that directly connects two routers, the path through the tunnel is almost always shorter than any path that goes through the real physical network.

The other way to look at the problem of recursive routing is to think about what the router has to do to a packet that it wants to send through the tunnel. It wraps this packet in the payload of a GRE packet and it puts the tunnel's destination address in the header of this GRE packet. Then it looks in its routing table to find out where to send this packet. If it finds that the best path is through the tunnel, then it must take this GRE packet and wrap it in the payload of a GRE packet whose destination address is the tunnel destination, and so on. This makes it difficult to deliver the original packet, so the router shuts down the tunnel interface to avoid having to stuff an infinite number of GRE headers onto the front of the packet.

There are two extremely simple solutions to this problem, but they aren't always applicable. You can use static routes to connect to the tunnel destination, which allows you to force the tunnel traffic to go the right way. Or you can prevent the routing protocol from passing through the tunnel either by using a separate IP address range or access lists. These two options are the first two examples in the solutions section of this recipe.

Note that we have used a specific host route for the destination IP address to ensure that it always uses the right path:

```
Router1(config)#ip route 172.22.1.2 255.255.255.255 172.25.1.1
```

The problem with this solution is that it might eliminate some of your network redundancy. For example, if there are several paths to the router that hold the tunnel's destination, using a static route like this might mean that your tunnel will fail if there is a topology change in the network that affects the manually selected path. In many cases, you can get around this problem by pointing the static route at a carefully selected downstream destination address. But then you run the risk that the router will learn about this downstream destination address through the tunnel, in which case we're back at square one.

The second simple solution is to simply exclude the tunnel from your routing protocol. In the example, we gave the tunnel an IP address that doesn't belong to the same address range as the source or destination addresses. This makes excluding the tunnel from the routing protocol relatively easy. By simply not including the `192.168.35.0/24` network in any of the EIGRP network commands, we prevent the tunnel from taking part in the routing protocol. We could also exclude the interface from the routing protocol using a distribute list. Please see Chapter 7 for more information about these EIGRP features.

However, sometimes these simple solutions are not appropriate. Some network topologies require that you use a routing protocol both inside and outside of the tunnel. For example, if you use VPNs to construct your WAN, through either a private or a public IP network, you will probably have to have both.

The best way to approach this type of situation is to start by ensuring that the ranges of IP addresses are distinct. For example, if the network that supports the tunnel uses public addressing, you could use private addressing for routes that need to be learned through the tunnel. Then you can apply a filter to prevent the routes for the supporting network from passing through the tunnel.

There are two ways to accomplish this. One is simply to use distinct routing protocols inside and outside of the tunnel, and not redistribute between the protocols. For example, the routing protocol outside of the tunnel could be BGP, while the tunneled network uses OSPF or EIGRP through the tunnel, or EIGRP and RIP as in the example above.

But another method is necessary if the two sets of routes use the same routing protocol, or if you need to redistribute. With distance vector-type interior routing protocols such as EIGRP and RIP, you can apply a route distribution filter to the tunnel interface to block the supporting network's routes. Note that EIGRP is much more sophisticated than a simple distance vector protocol like RIP. But this kind of route filtering is not possible with link state protocols such as OSPF, so this is one place where EIGRP's distance vector roots come in handy.

In the example, we have shown how to do route filtering using a prefix list with EIGRP. This will

permit only the `192.168.0.0/16` range of IP addresses to pass through the tunnel, while information about the `172.22.0.0/16` and `172.25.0.0/16` networks that form the support network is never learned through the tunnel. You should apply this type of filter to both ends of the tunnel:

```
Router11(config)#ip prefix-list TUNNELROUTES seq 10 permit  192.168.0.0/16 ge 17
Router1(config)#router eigrp 55
Router1(config-router)#network 172.22.0.0
Router1(config-router)#network 172.25.0.0
Router1(config-router)#distribute-list prefix TUNNELROUTES out Tunnel1
```

For more information on EIGRP route distribution filters, please see Chapter 7. For more information on prefix lists, you can refer to either Chapter 7 or Chapter 9.

12.3.4 See Also

Recipe 12.1; Chapter 7 and Chapter 9

Top

Recipe 12.4 Viewing Tunnel Status

12.4.1 Problem

You want to check the status of a tunnel.

12.4.2 Solution

You can look at the attributes for a tunnel with the *show interface* command:

```
Router1#show interface Tunnel5
```

The easiest way to determine if a tunnel is operational is simply to use a ping test to either the send ICMP packets through the tunnel or to its destination address:

```
Router1#ping 192.168.66.6
Router1#ping 172.22.1.4
```

12.4.3 Discussion

You can use the standard show interface command on a tunnel interface to see a considerable amount of useful information about it:

```
Router1#show interface Tunnel5
Tunnel5 is up, line protocol is up
  Hardware is Tunnel
   Internet address is 192.168.66.5/30
  MTU 1514 bytes, BW 9 Kbit, DLY 500000 usec,
      reliability 255/255, txload 1/255, rxload 1/255
  Encapsulation TUNNEL, loopback not set
  Keepalive not set
   Tunnel source 172.22.1.3, destination 172.22.1.4
  Tunnel protocol/transport GRE/IP, key disabled, sequencing disabled
  Checksumming of packets disabled,  fast tunneling enabled
   Last input 1d19h, output 00:00:06, output hang never
  Last clearing of "show interface" counters never
   Input queue: 0/75/0/0 (size/max/drops/flushes); Total output drops: 79
  Queueing strategy: fifo
  Output queue: 0/0 (size/max)
  5 minute input rate 0 bits/sec, 0 packets/sec
  5 minute output rate 0 bits/sec, 0 packets/sec
     2536 packets input, 1386605 bytes, 0 no buffer
```

```
     Received 0 broadcasts, 0 runts, 0 giants, 0 throttles
    0 input errors, 0 CRC, 0 frame, 0 overrun, 0 ignored, 0 abort
     23235 packets output, 2036436 bytes, 0 underruns
     0 output errors, 0 collisions, 0 interface resets
     0 output buffer failures, 0 output buffers swapped out
Router1#
```

As you can see from this output, the *show interface* command tells you what the tunnel's source and destination IP addresses are. You can see the input and output rate, as well as the total number of packets and bytes both sent and received on this tunnel interface. The output also shows that we are using the default GRE tunnel protocol and not enabled checksums or keepalives on this tunnel.

There is only one serious problem with this output. Because we have not enabled keepalives, we discussed in Recipe 12.1, the *show interface* command will almost always show the tunnel interface as being in an up state. As we mentioned in Recipe 12.1, the router will temporarily bring the tunnel interface down in response to recursive routing situations, and you can also use the *shutdown* command to disable a tunnel as you would with any other interface. However, usually the tunnel interface will appear to be in an up state, even if the router can't reach the tunnel destination router.

If you are running an IOS level that supports keepalives on tunnels, you can enable that feature. Then the show interface command will give a more realistic view of the tunnel's status. But, without that feature, the easiest way to see if a tunnel is working is to simply ping through it:

```
Router1#ping 192.168.66.6

Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 192.168.66.6, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 12/14/20 ms
Router1#
```

Alternatively, you can ping the destination IP address of the tunnel:

```
Router1#ping 172.22.1.4

Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 192.168.66.6, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 12/14/20 ms
Router1#
```

12.4.4 See Also

Recipe 12.1

Top

Recipe 12.5 Creating an EncryptedRouter-to-Router VPN

12.5.1 Problem

You want to create an encrypted VPN through the Internet connecting two routers using pre-shared keys.

12.5.2 Solution

In this example, we show how to use IPSec to encrypt traffic from one router to another through a GRE tunnel. Here is the configuration of the first router:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#crypto isakmp policy 10
Router1(config-isakmp)#encryption 3des
Router1(config-isakmp)#authentication pre-share
Router1(config-isakmp)#group 2
Router1(config-isakmp)#exit
Router1(config)#crypto isakmp key TUNNELKEY01 address 172.22.1.4 no-xauth
Router1(config)#crypto ipsec transform-set TUNNEL-TRANSFORM ah-sha-hmac esp-3des esp-
sha-hmac
Router1(cfg-crypto-trans)#mode transport
Router1(cfg-crypto-trans)#exit
Router1(config)#crypto map TUNNELMAP 10 ipsec-isakmp
Router1(config-crypto-map)#set peer 172.22.1.4
Router1(config-crypto-map)#set transform-set TUNNEL-TRANSFORM
Router1(config-crypto-map)#match address 116
Router1(config-crypto-map)#exit
Router1(config)#access-list 116 permit gre host 172.22.1.3 host 172.22.1.4
Router1(config)#interface Tunnel5
Router1(config-if)#ip address 192.168.66.5 255.255.255.252
Router1(config-if)#tunnel source 172.22.1.3
Router1(config-if)#tunnel destination 172.22.1.4
Router1(config-if)#exit
Router1(config)#interface FastEthernet0/1
Router1(config-if)#ip address 172.22.1.3 255.255.255.0
Router1(config-if)#crypto map TUNNELMAP
Router1(config-if)#end
Router1#
```

Here is the corresponding configuration for the other router:

```
Router2#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router2(config)#crypto isakmp policy 10
```

```
Router2(config-isakmp)#encryption 3des
Router2(config-isakmp)#authentication pre-share
Router2(config-isakmp)#group 2
Router2(config-isakmp)#exit
Router2(config)#crypto isakmp key TUNNELKEY01 address 172.22.1.3 no-xauth
Router2(config)#crypto ipsec transform-set TUNNEL-TRANSFORM ah-sha-hmac esp-3des esp-
sha-hmac
Router2(cfg-crypto-trans)#mode transport
Router2(cfg-crypto-trans)#exit
Router2(config)#crypto map TUNNELMAP 10 ipsec-isakmp
Router2(config-crypto-map)#set peer 172.22.1.3
Router2(config-crypto-map)#set transform-set TUNNEL-TRANSFORM
Router2(config-crypto-map)#match address 116
Router2(config-crypto-map)#exit
Router2(config)#access-list 116 permit gre host 172.22.1.4 host 172.22.1.3
Router2(config)#interface Tunnel5
Router2(config-if)#ip address 192.168.66.6 255.255.255.252
Router2(config-if)#tunnel source 172.22.1.4
Router2(config-if)#tunnel destination 172.22.1.3
Router2(config-if)#exit
Router2(config)#interface FastEthernet1/0
Router2(config-if)#ip address 172.22.1.4 255.255.255.0
Router2(config-if)#crypto map TUNNELMAP
Router2(config-if)#end
Router2#
```

12.5.3 Discussion

There are several steps to these configurations, but they are the same on both routers. The first step is to create an appropriate key exchange policy using ISAKMP. The following set of commands defines the policy with priority 10. When ISAKMP negotiates the Security Association (SA) parameters, it starts with the highest priority and goes to the lowest. The highest possible priority value is 10,000:

```
Router1(config)#crypto isakmp policy 10
Router1(config-isakmp)#encryption 3des
Router1(config-isakmp)#authentication pre-share
Router1(config-isakmp)#group 2
```

This policy uses 168-bit Triple DES encryption, pre-shared authentication keys, and group 2 (1024-bit) for DH exchange. If we did not configure this, the routers would have to resort to the default parameters, which are 56-bit DES encryption, Rivest Shamir Adleman (RSA) signatures for authentication and DH group 1 (768-bit). You can see the available policies on a router with the *show crypto isakmp policy* command:

```
Router1#show crypto isakmp policy
Protection suite of priority 10
        encryption algorithm:   Three key triple DES
        hash algorithm:         Secure Hash Standard
        authentication method:  Pre-Shared Key
```

```
        Diffie-Hellman group:   #2 (1024 bit)
        lifetime:               86400 seconds, no volume limit
Default protection suite
        encryption algorithm:   DES - Data Encryption Standard (56 bit keys).
        hash algorithm:         Secure Hash Standard
        authentication method:  Rivest-Shamir-Adleman Signature
        Diffie-Hellman group:   #1 (768 bit)
        lifetime:               86400 seconds, no volume limit
```

We could have also adjusted the hash algorithm and the lifetime of a particular SA as follows:

```
Router1(config)#crypto isakmp policy 20
Router1(config-isakmp)#hash md5
Router1(config-isakmp)#lifetime 600
```

This policy uses the somewhat less secure but faster MD5 hash algorithm and reduces the SA lifetime to 600 seconds (10 minutes). The default hash algorithm is the standard IPSec SHA, and the default lifetime is 86,400 seconds (24 hours). Reducing the lifetime forces the routers to renegotiate the various SA parameters, including encryption keys, more frequently. This frequent renegotiation improves security, but at the expense of higher router CPU utilization and possible delays during the renegotiation process.

Because we have configured the routers to use pre-shared keys in this policy, we need to define this initial key with the *crypto isakmp key* command:

```
Router1(config)#crypto isakmp key TUNNELKEY01 address 172.22.1.4 no-xauth
```

As you can see, this sets this key only for one IP address, which is the address of the other router. We have included the *no-xauth* option on the command line to explicitly disable IKE Extended Authentication (XAuth) on the routers, which is not necessary when the peer is another router. ISAKMP can work with either IP addresses or hostnames to identify devices. So we could have specified this command like this instead:

```
Router1(config)#crypto isakmp key TUNNELKEY01 hostname Router2.oreilly.com no-xauth
```

However, to do this, we would also have needed to ensure that the remote device used its hostname when declaring its ISAKMP identity:

```
Router2(config)#crypto isakmp identity hostname
```

We avoided this extra complication by simply using IP addresses, which is the default behavior.

There are several useful commands for looking at the ISAKMP functions on your router. The first is *show crypto isakmp key* , which lists all of the pre-shared keys that are available:

```
Router1#show crypto isakmp key
Hostname/Address        Preshared Key
172.22.1.4              TUNNELKEY01
Router1#
```

Note that this doesn't mean that there is an active SA using this key, merely that the key is available if required. If you want to see information on active ISAKMP SAs, you should use the following command:

```
Router1#show crypto isakmp sa
dst             src             state          conn-id    slot
172.22.1.4      172.22.1.3      QM_IDLE              1       0

Router1#
```

In this case, you can see that there is an active connection between the two routers shown in the example. The connection ID for this particular SA is shown in the conn-id column. You can use this ID number to clear the SA and force the routers to renegotiate as follows:

```
Router1#clear crypto isakmp 1
Router1#show crypto isakmp sa
dst             src             state          conn-id    slot
172.22.1.4      172.22.1.3      MM_NO_STATE          1       0   (deleted)

Router1#
```

This particular SA is now in a deleted state, as the routers begin to renegotiate their ISAKMP parameters. A short time later, they will re-establish a new SA.

The next part of the router configuration defines the IPSec *transform set* , and gives it the name `TUNNEL-TRANSFORM` , to distinguish it from other transform sets that we might want to use for other purposes:

```
Router1(config)#crypto ipsec transform-set TUNNEL-TRANSFORM ah-sha-hmac esp-3des esp-
sha-hmac
Router1(cfg-crypto-trans)#mode transport
```

A transform is simply the operation that IPSec will perform on any matching data packets. There are several possible transforms, which are listed in Table 12-2 .

Table 12-2. IPSec transform set options

Transform type	Transform name	Description
Compression	comp-lzs	Compress using Lempel Ziv Stac algorithm
Authentication Header (AH)	ah-md5-hmac	Authenticate using MD5 algorithm
	ah-sha-hmac	Authenticate using SHA algorithm

Transform type	Transform name	Description
Encapsulating Security Payload (ESP)	esp-des	Encrypt using 56-bit DES
	esp-3des	Encrypt using 168-bit DES
	esp-null	No encryption
ESP with authentication	esp-md5-hmac	Encrypt, and use MD5 for authentication
	esp-sha-hmac	Encrypt, and use SHA for authentication

In the above example, we chose to combine the more secure 168-bit DES encryption with the more reliable SHA authentication system for maximum security on both the AH and ESP portions of the packet. As we said earlier, many of the combinations are not possible (for example, you cannot combine 56-bit DES with 168-bit DES, as it doesn't make sense). The router will prevent you from entering impossible combinations.

It's also worth mentioning that you can use IPSec to compress the IP packet payload. This is not commonly done, though, because there are problems with combining encryption with compression.

While the authors of the IPSec RFCs argue eloquently for the separate usefulness of authentication and encryption, in practice we believe that if your traffic is important enough for one, you should do both. The extra configuration on the router is minimal, and if your router's CPU can't easily handle the load of both encrypting and authenticating, it is probably not the right router for the job. So, if you are going to either authenticate or encrypt your traffic, we recommend using both together for added security.

In this transform set, we have also instructed the router to use IPSec transport mode:

```
Router1(cfg-crypto-trans)#mode transport
```

By default, IPSec connections will use tunnel mode, which means that the two devices will set up their own tunnel for IPSec to use. This actually uses the IP-in-IP tunnel protocol that we mentioned in the introduction to this chapter. However, in this example we want to use a GRE tunnel between the routers instead, and simply authenticate and encrypt the GRE packets, which requires transport mode.

The main reasons for using GRE tunnels instead of IPSec's native tunnel mode are simplicity and flexibility. Using a GRE tunnel between these routers allows us to take advantage of some of the useful GRE features if we want them. For example, we can easily use a GRE tunnel to pass other protocols such as IPX or AppleTalk, and we can use the techniques discussed in Recipe 12.3 to pass routing protocol traffic through the encrypted tunnel. The GRE tunnel also makes debugging much easier, as we can simply disable the encryption and ping through the tunnel or ping the tunnel destination addresses to verify connectivity without the complications of authentication and encryption. If the other end of this tunnel was a workstation instead of a router, however (as in Recipe 12.6), we would have to use tunnel

mode.

The next step is to define a crypto map that combines all of these elements. The following set of commands defines a map called TUNNELMAP . The number following this name is a sequence number, similar to the route map sequence numbers that we discussed in Chapter 6 , Chapter 7 and Chapter 8 . This allows you to associate many peers with a single router interface by creating several different map clauses with different sequence numbers, all associated with the same map.

The keyword *ipsec-isakmp* on the end of the *crypto map* command tells the router that this map will apply to IPSec connections that use ISAKMP for key management. You could also specify *ipsec-manual* if you wanted to do the key management manually. We don't generally recommend manual key management because it is so much trouble to get right, while ISAKMP automates most of the work for you:

```
Router1(config)#crypto map TUNNELMAP 10 ipsec-isakmp
Router1(config-crypto-map)#set peer 172.22.1.4
Router1(config-crypto-map)#set transform-set TUNNEL-TRANSFORM
Router1(config-crypto-map)#match address 116
Router1(config-crypto-map)#exit
Router1(config)#access-list 116 permit gre host 172.22.1.3 host 172.22.1.4
```

The crypto map defines an IPSec peer device by its IP address. If you are using hostnames instead of IP addresses (as discussed earlier in this recipe), you should specify the peer's hostname instead of an IP address. The map also selects the appropriate transform set, and matches on a particular set of IP addresses, defined in this case by access-list 116.

The access list tells IPSec what packets it should apply this transform set to. In this case, we specify a source IP address of `172.22.1.3` , which is the IP address of the tunnel source, and `172.22.1.4` , which is the tunnel's destination address. And because of the *gre* keyword, this access list will only match GRE tunnel packets with these source and destination addresses.

On the other router, the peer address is `172.22.1.3` , and the access list reverses the source and destination addresses:

```
Router2(config)#access-list 116 permit gre host 172.22.1.4 host 172.22.1.3
```

Then, with all of the IPSec and ISAKMP configuration in place, we can finally create the tunnel and turn on the encryption. The tunnel configuration is similar to what we used in Recipe 12.1

```
Router1(config)#interface Tunnel5
Router1(config-if)#ip address 192.168.66.5 255.255.255.252
Router1(config-if)#tunnel source 172.22.1.3
Router1(config-if)#tunnel destination 172.22.1.4
Router1(config-if)#exit
Router1(config)#interface FastEthernet0/1
Router1(config-if)#ip address 172.22.1.3 255.255.255.0
Router1(config-if)#crypto map TUNNELMAP
```

It's extremely important to notice that we have applied the crypto map to the interface that will be receiving the GRE packets, and not to the tunnel itself. This is because IPSec is encrypting the GRE tunnel packets, rather than the payload of those packets. For one thing, the GRE tunnel's payload is not necessarily an IP packet. However, even when they are IP packets, the source and destination IP addresses of the GRE payload could be devices somewhere behind the router. This breaks the essential requirement for IPSec's transport mode, which is that the source and destination IP addresses must be the devices themselves. So the only way you could successfully apply the crypto map to the tunnel interface would be by using an IPSec tunnel inside of the GRE tunnel, which would not be very efficient.

You can see that the encryption is working properly by looking at the output of the following command on either router:

```
Router2#show crypto engine connections active

  ID Interface       IP-Address      State  Algorithm           Encrypt  Decrypt
   1 FastEthernet1/0 172.22.1.4      set    HMAC_SHA+3DES_56_C        0        0
2002 FastEthernet1/0 172.22.1.4      set    HMAC_SHA+3DES_56_C        0      407
2003 FastEthernet1/0 172.22.1.4      set    HMAC_SHA+3DES_56_C     1019        0

Router2#
```

This shows that the router has received and decrypted 407 encrypted packets from the peer that we defined, and it has sent 1,019. It also shows that we are using the SHA hash algorithm for authentication and Triple DES for encryption in the Algorithm column.

12.5.4 See Also

Recipe 12.1 ; Recipe 12.3

Top

Recipe 12.6 Generating RSA Keys

12.6.1 Problem

You want to create a shareable RSA key for authentication or encryption.

12.6.2 Solution

First, you must create the keys on both devices. We recommend using at least 1024- bit keys in production networks:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#crypto key generate rsa
The name for the keys will be: Router1.oreilly.com
Choose the size of the key modulus in the range of 360 to 2048 for your
  General Purpose Keys. Choosing a key modulus greater than 512 may take
  a few minutes.

How many bits in the modulus [512]: 1024
Generating RSA keys ...
[OK]

Router1(config)#end
Router1#show crypto key mypubkey rsa
% Key pair was generated at: 01:19:45 EST Mar 1 2003
Key name: Router1.oreilly.com
 Usage: General Purpose Key
 Key Data:
  30819F30 0D06092A 864886F7 0D010101 05000381 8D003081 89028181 00E68338
  D561B2D1 7B8B75D6 7B34F6AF 1710B00B 5B6E9E8D D7183BE6 F08A6342 054EADFC
  B764DF9C 4592B891 522727F2 14233B47 8F757134 24F03DB3 833C5988 312B11E9
  FB6E0E20 4579C0A4 F2062353 4F1C8CE4 410EE57B 9FCEE784 DA7E3852 408E9742
  2584DF56 67293F3F F76B6A96 C4D518FB 1A0114BF E2449838 BE5794E2 37020301 0001
% Key pair was generated at: 01:19:52 EST Mar 1 2003
Key name: Router1.oreilly.com.server
 Usage: Encryption Key
 Key Data:
  307C300D 06092A86 4886F70D 01010105 00036B00 30680261 00BD928A BD5637E6
  2265621C 3AC57138 911CA27D 11F40AA1 E657EA26 6EBF654C 952A3319 D421A33C
  E2ECA87E CD7E050C 8A8FE64D B73954EA BF2ED639 BC6A8F74 5B9550EA 4119E796
  A97430E2 4B1BF7D3 ED1469FF AEA83690 A0FEA871 BBFBE8AD 19020301 0001
Router1#
```

Then you can use cut and paste to copy this manually generated key into the other device:

```
Router2#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router2(config)#crypto key pubkey-chain rsa
Router2(config-pubkey-chain)#addressed-key 192.168.99.1
Router2(config-pubkey-key)#address 192.168.99.1
Router2(config-pubkey-key)#key-string
Enter a public key as a hexidecimal number ....

Router2(config-pubkey)#30819F30 0D06092A 864886F7 0D010101 05000381 8D003081 89028181
00E68338
Router2(config-pubkey)#D561B2D1 7B8B75D6 7B34F6AF 1710B00B 5B6E9E8D D7183BE6 F08A6342
054EADFC
Router2(config-pubkey)#B764DF9C 4592B891 522727F2 14233B47 8F757134 24F03DB3 833C5988
312B11E9
Router2(config-pubkey)#FB6E0E20 4579C0A4 F2062353 4F1C8CE4 410EE57B 9FCEE784 DA7E3852
408E9742
Router2(config-pubkey)#2584DF56 67293F3F F76B6A96 C4D518FB 1A0114BF E2449838 BE5794E2
37020301 0001
Router2(config-pubkey)#quit
Router2(config-pubkey-key)#exit
Router2(config-pubkey-chain)#exit
Router2(config)#end
Router2#show crypto key pubkey-chain rsa address 192.168.99.1
Key address: 192.168.99.1
 Usage: General Purpose Key
 Source: Manually entered
 Data:
  30819F30 0D06092A 864886F7 0D010101 05000381 8D003081 89028181 00E68338
  D561B2D1 7B8B75D6 7B34F6AF 1710B00B 5B6E9E8D D7183BE6 F08A6342 054EADFC
  B764DF9C 4592B891 522727F2 14233B47 8F757134 24F03DB3 833C5988 312B11E9
  FB6E0E20 4579C0A4 F2062353 4F1C8CE4 410EE57B 9FCEE784 DA7E3852 408E9742
  2584DF56 67293F3F F76B6A96 C4D518FB 1A0114BF E2449838 BE5794E2 37020301 0001


Router2#
```

12.6.3 Discussion

When you use the *crypto key generate* command to create new keys, the router must delete any existing keys:

```
Router1(config)#crypto key generate rsa
The name for the keys will be: Router1.oreilly.com
% You already have RSA keys defined for Router1.oreilly.com.
% Do you really want to replace them? [yes/no]: yes
Choose the size of the key modulus in the range of 360 to 2048 for your
  General Purpose Keys. Choosing a key modulus greater than 512 may take
  a few minutes.

How many bits in the modulus [512]: 1024
Generating RSA keys ...
```

```
[OK]

Router1(config)#
```

During key generation, any services on the router that is currently using these keys will be temporarily disabled. Key generation can take a considerable length of time, depending on the model of router and the size of the key modulus. We have seen a low-end access router take as long as an hour to generate a key with a very large modulus for greater security. During this time, the router's CPU load was extremely high. So we urge caution when using this command.

You can remove existing keys with the *crypto key zeroize* command:

```
Router1(config)#crypto key zeroize rsa
% Keys to be removed are named Router1.oreilly.com.
Do you really want to remove these keys? [yes/no]: yes
Router1(config)#
```

If the router has any services that are using the deleted keys, it will automatically disable them until you generate new keys.

You can also generate special usage keys as follows:

```
Router1(config)#crypto key generate rsa usage-keys
The name for the keys will be: Router1.oreilly.com
% You already have RSA keys defined for Router1.oreilly.com.
% Do you really want to replace them? [yes/no]: yes
Choose the size of the key modulus in the range of 360 to 2048 for your
  Signature Keys. Choosing a key modulus greater than 512 may take
  a few minutes.

How many bits in the modulus [512]: 1024
Generating RSA keys ...
[OK]
Choose the size of the key modulus in the range of 360 to 2048 for your
  Encryption Keys. Choosing a key modulus greater than 512 may take
  a few minutes.

How many bits in the modulus [512]: 1024
Generating RSA keys ...
[OK]

Router1(config)#
```

This command creates separate authentication signature and encryption keys. You can look at the public keys with the *show crypto key* command:

```
Router1#show crypto key mypubkey rsa
% Key pair was generated at: 01:29:04 EST Mar 1 2003
Key name: Router1.oreilly.com
 Usage: Signature Key
```

```
 Key Data:
  30819F30 0D06092A 864886F7 0D010101 05000381 8D003081 89028181 00AAED98
  0E454C8F ED9DB93E 312B00BD FF561C49 5480344A 094F0EA8 0D994051 AC627CF2
  5FA7F802 DB0A1206 4EB8F8E5 122C9B2D 0F3A20D8 C0E90280 D4F6518A 9C6C2E48
  A570D05A AE2881CA B9366990 931C4A7E EDC6B352 13815B91 3A02B44E 4655DE6D
  1CB5AB35 058B60AA 4639B696 A8EE735E DA15B300 B8A0CE51 7C42B73A 53020301 0001
% Key pair was generated at: 01:29:11 EST Mar 1 2003
Key name: Router1.oreilly.com
 Usage: Encryption Key
 Key Data:
  30819F30 0D06092A 864886F7 0D010101 05000381 8D003081 89028181 00D18F99
  EC2A5754 C1FEF911 E16BFD80 6C3E9517 42716B78 99692618 B57B529B A9C19B23
  6D4BF3CE 39728DEF 2B3D10F9 3DABBDFD 8CAB09F7 0A56768C 053BB4AF 7F224E44
  FA341851 10152A86 28C2084F C13E0738 4C478BED 9960E229 CB112077 097F3DC9
  DD40D109 0A513D31 FF0FD51D B3515CEA F81738B6 5BB02FF6 812A01AC F7020301 0001
% Key pair was generated at: 01:29:14 EST Mar 1 2003
Key name: Router1.oreilly.com.server
 Usage: Encryption Key
 Key Data:
  307C300D 06092A86 4886F70D 01010105 00036B00 30680261 00B43311 D047EFBC
  314C57DB 93F3E755 5CEBF4B5 D0258169 6DAC695B A0F5DA35 C6C7B106 C2BB7863
  0201B68A 7C2F3313 47223065 BDF84692 BF974F2E E4037D5D C976DB3A 231D2603
  6DE8CDCE 8EAD613E 5C984091 55A6B0F5 920E285B 6E4ED34E 31020301 0001
Router1#
```

As you can see, the router now has a signature key and an encryption key where it previously had only a general purpose key. However, it is important to remember that this is only the public key. There is also a corresponding private key that you cannot view on the router. The router keeps this key in its NVRAM storage and sets file permissions so nobody can read it. The private key is what the router uses to encrypt things that it sends. The public key can decrypt anything encrypted with the private key. Every device that this router shares encrypted information with will need a copy of the public key, but the private key is secret.

As a side effect of this, the public key provides an excellent authentication system. If the device's public key successfully decrypts a message, then you know that this message must have been encrypted with that device's private key. And, consequently, if the private key really is private, the message must actually have been sent by that device.

When you use these keys on routers, we highly recommend using the cut-and-paste feature on your terminal, rather than trying to type all of this in manually. A single typographical error in this sequence will make the key useless. Note, however, that there is an inherent security risk in copying and pasting a key like this over a network. If you are using an insecure protocol like Telnet, the packet can be intercepted, and the key information is easily extracted. So you should avoid doing this over untrusted networks.

In Recipe 12.7 we will show an example of how to use RSA keys.

12.6.4 See Also

Recipe 12.7

Top

Recipe 12.7 Creating a Router-to-Router VPN with RSA Keys

12.7.1 Problem

You want to create an encrypted VPN between two routers using RSA keys.

12.7.2 Solution

As in Recipe 12.3 , we will use IPSec transport mode and a GRE tunnel for this encrypted router-to-router connection:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#crypto key pubkey-chain rsa
Router1(config-pubkey-chain)#addressed-key 172.22.1.4
Router1(config-pubkey-key)#address 172.22.1.4
Router1(config-pubkey-key)#key-string
Enter a public key as a hexidecimal number ....

Router1(config-pubkey)#305C300D 06092A86 4886F70D 01010105 00034B00 30480241 00B8120C
AC2C5AAC
Router1(config-pubkey)#ADAD378D A5A1F140 2EB9A66A DD6FF2A9 7DD47692 5CDE4732 E2C9EDDA
52809BE0
Router1(config-pubkey)#D60A5A34 CDD7DC31 DA7F9590 849F142E 26C6F130 0A2E4491 65020301
0001
Router1(config-pubkey)#quit
Router1(config-pubkey-key)#exit
Router1(config-pubkey-chain)#exit
Router1(config)#crypto isakmp policy 100
Router1(config-isakmp)#encryption 3des
Router1(config-isakmp)#authentication rsa-encr
Router1(config-isakmp)#group 2
Router1(config-isakmp)#exit
Router1(config)#crypto ipsec transform-set TUNNEL-TRANSFORM ah-sha-hmac esp-3des esp-
sha-hmac
Router1(cfg-crypto-trans)#mode transport
Router1(cfg-crypto-trans)#exit
Router1(config)#crypto map TUNNEL-RSA 10 ipsec-isakmp
Router1(config-crypto-map)#set peer 172.22.1.4
Router1(config-crypto-map)#set transform-set TUNNEL-TRANSFORM
Router1(config-crypto-map)#match address 116
Router1(config-crypto-map)#exit
Router1(config)#access-list 116 permit gre host 172.22.1.3 host 172.22.1.4
Router1(config)#interface Tunnel5
Router1(config-if)#ip address 192.168.66.5 255.255.255.252
Router1(config-if)#tunnel source 172.22.1.3
```

```
Router1(config-if)#tunnel destination 172.22.1.4
Router1(config-if)#exit
Router1(config)#interface FastEthernet0/1
Router1(config-if)#ip address 172.22.1.3 255.255.255.0
Router1(config-if)#crypto map TUNNEL-RSA
Router1(config-if)#end
Router1#
```

Here is the corresponding configuration for the other router:

```
Router2#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router2(config)#crypto key pubkey-chain rsa
Router2(config-pubkey-chain)#addressed-key 172.22.1.3
Router2(config-pubkey-key)#address 172.22.1.3
Router2(config-pubkey-key)#key-string
Enter a public key as a hexidecimal number ....

Router2(config-pubkey)#305C300D 06092A86 4886F70D 01010105 00034B00 30480241 00DA2810
6627212B
Router2(config-pubkey)#7234CC4A 91BCB4CC 7985AD4B 884E4653 9E533422 A34A4011 E7402B56
7DCE7A33
Router2(config-pubkey)#7407C6DE 995D2EBD E9F2C29D B4EEB962 61B9CF3B 05D903FC 57020301
0001
Router2(config-pubkey)#quit
Router2(config-pubkey-key)#exit
Router2(config-pubkey-chain)#exit
Router2(config)#crypto isakmp policy 100
Router2(config-isakmp)#encryption 3des
Router2(config-isakmp)#authentication rsa-encr
Router2(config-isakmp)#group 2
Router2(config-isakmp)#exit
Router2(config)#crypto ipsec transform-set TUNNEL-TRANSFORM ah-sha-hmac esp-3des esp-
sha-hmac
Router2(cfg-crypto-trans)#mode transport
Router2(cfg-crypto-trans)#exit
Router2(config)#crypto map TUNNEL-RSA 10 ipsec-isakmp
Router2(config-crypto-map)#set peer 172.22.1.3
Router2(config-crypto-map)#set transform-set TUNNEL-TRANSFORM
Router2(config-crypto-map)#match address 116
Router2(config-crypto-map)#exit
Router2(config)#access-list 116 permit gre host 172.22.1.4 host 172.22.1.3
Router2(config)#interface Tunnel5
Router2(config-if)#ip address 192.168.66.6 255.255.255.252
Router2(config-if)#tunnel source 172.22.1.4
Router2(config-if)#tunnel destination 172.22.1.3
Router2(config-if)#exit
Router2(config)#interface FastEthernet1/0
Router2(config-if)#ip address 172.22.1.4 255.255.255.0
Router2(config-if)#crypto map TUNNEL-RSA
Router2(config-if)#end
Router2#
```

12.7.3 Discussion

This recipe is similar to Recipe 12.3 , except that here we use RSA keys for authentication and encryption instead of pre-shared keys. This technique is more secure but more time consuming to configure.

The first step is to create a set of RSA encryption keys using the methods discussed in Recipe 12.6We took the keys that we generated in this way and entered them into the router configurations. So, for example, we created the key on Router1 as follows:

```
Router1(config)#crypto key generate rsa
The name for the keys will be: Router1.oreilly.com
% You already have RSA keys defined for Router1.oreilly.com.
% Do you really want to replace them? [yes/no]: yes
Choose the size of the key modulus in the range of 360 to 2048 for your
  General Purpose Keys. Choosing a key modulus greater than 512 may take
  a few minutes.

How many bits in the modulus [512]:
Generating RSA keys ...
[OK]

Router1(config)#exit
Router1#show crypto key mypubkey rsa
% Key pair was generated at: 11:25:55 EST Jan 26 2003
Key name: Router1.oreilly.com
 Usage: General Purpose Key
 Key Data:
  305C300D 06092A86 4886F70D 01010105 00034B00 30480241 00DA2810 6627212B
  7234CC4A 91BCB4CC 7985AD4B 884E4653 9E533422 A34A4011 E7402B56 7DCE7A33
  7407C6DE 995D2EBD E9F2C29D B4EEB962 61B9CF3B 05D903FC 57020301 0001
% Key pair was generated at: 11:26:01 EST Jan 26 2003
Key name: Router1.oreilly.com.server
 Usage: Encryption Key
 Key Data:
  307C300D 06092A86 4886F70D 01010105 00036B00 30680261 00A3603A 58941769
  EF93B43D C89AC7CF 2A6DA0D5 F72BCFF8 D9EEDDD2 B0CE9A8E B4BAFD2D 805A4D8F
  969A5AE3 5F4F8252 744A0834 B4BA24B9 BC7E4522 2345F081 587BD1A8 309B03F4
  A70F2373 2AB6CEE5 736F6D61 F64A94A6 30CE253F BEB8330B FF020301 0001
Router1#
```

Note that in this example we used the default 512-bit key. However, in production networks, we recommend using 1024-bit keys or higher.

Then we took the general purpose key from this output and entered it into the other router as follows:

```
Router2(config)#crypto key pubkey-chain rsa
Router2(config-pubkey-chain)#addressed-key 172.22.1.3
Router2(config-pubkey-key)#address 172.22.1.3
```

```
Router2(config-pubkey-key)#key-string
Enter a public key as a hexidecimal number ....

Router2(config-pubkey)#305C300D 06092A86 4886F70D 01010105 00034B00 30480241 00DA2810
6627212B
Router2(config-pubkey)#7234CC4A 91BCB4CC 7985AD4B 884E4653 9E533422 A34A4011 E7402B56
7DCE7A33
Router2(config-pubkey)#7407C6DE 995D2EBD E9F2C29D B4EEB962 61B9CF3B 05D903FC 57020301
0001
Router2(config-pubkey)#quit
Router2(config-pubkey-key)#exit
Router2(config-pubkey-chain)#exit
```

We then repeated the procedure on the other router.

With the keys in place, we proceeded to tell the routers how to use these keys to create an IPSec connection. Even though we are using a manually entered key, the two routers still need to use ISAKMP. The important difference between this example and the one in Recipe 12.3 is that here we are using RSA authentication keys. So we need to tell the routers to use this key method in the ISAKMP policy:

```
Router1(config)#crypto isakmp policy 100
Router1(config-isakmp)#encryption 3des
Router1(config-isakmp)#authentication rsa-encr
Router1(config-isakmp)#group 2
```

After that, the remainder of the configuration is essentially identical to what we showed in Recipe 12.3.

12.7.4 See Also

Recipe 12.3 ; Recipe 12.6

Top

Recipe 12.8 Creating a VPN Between a Workstation and a Router

12.8.1 Problem

You want to make a VPN from a remote workstation to a router.

12.8.2 Solution

There are several steps to configuring a router to accept IPSec VPN connections from remote PCs. The following discussion doesn't include requirements for the PC's software configuration, just the router's configuration. You should refer to the software vendor's documentation for information about configuring the workstation software:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#aaa new-model
Router1(config)#aaa authentication login default group tacacs+
Router1(config)#aaa authentication enable default group tacacs+
Router1(config)#tacacs-server host 172.25.1.1
Router1(config)#tacacs-server key COOKBOOK
Router1(config)#crypto isakmp policy 10
Router1(config-isakmp)#encryption 3des
Router1(config-isakmp)#authentication pre-share
Router1(config-isakmp)#group 2
Router1(config-isakmp)#exit
Router1(config)#crypto ipsec transform-set VPN-TRANSFORMS ah-sha-hmac esp-sha-hmac
esp-3des
Router1(cfg-crypto-trans)#mode tunnel
Router1(cfg-crypto-trans)#exit
Router1(config)#crypto dynamic-map VPN-USER-MAP 50
Router1(config-crypto-map)#description A dynamic crypto map for VPN users
Router1(config-crypto-map)#match address 115
Router1(config-crypto-map)#set transform-set VPN-TRANSFORMS
Router1(config-crypto-map)#exit
Router1(config)#access-list 115 deny any 224.0.0.0 35.255.255.255
Router1(config)#access-list 115 deny any 172.25.1.255 0.0.0.0
Router1(config)#access-list 115 permit any any
Router1(config)#crypto map CRYPTOMAP 10 ipsec-isakmp dynamic VPN-USER-MAP
Router1(config)#interface FastEthernet0/1
Router1(config-if)#ip address 172.25.1.5 255.255.255.0
Router1(config-if)#crypto map CRYPTOMAP
Router1(config-if)#exit
Router1#
```

12.8.3 Discussion

The first few lines in this example are the *aaa* and *tacacs-server* commands, which are described in more detail in Chapter 4. This simply sets up username authentication for all incoming VPN connections, and allows you to get these authentication credentials from a central server running the TACACS+ protocol.

We are using AAA and TACACS+ in this configuration to supply the pre-shared keys that ISAKMP will use to set up its SA for this VPN. This is similar to the use of pre-shared keys in Recipe 12.3, but here we expect to have a large number of remote VPN users, so it is administratively easier if we manage them from the TACACS+ server instead of on the router.

Then we set up the ISAKMP policy as follows:

```
Router1(config)#crypto isakmp policy 10
Router1(config-isakmp)#encryption 3des
Router1(config-isakmp)#authentication pre-share
Router1(config-isakmp)#group 2
```

This defines the policy for authentication and encryption keys, and is identical to the ISAKMP policy we used in Recipe 12.3. We selected these particular policy parameters because they are required for the Cisco Easy VPN Remote software. If you are using different client software, you may need to use different settings. This policy is also identical to the one we used in Recipe 12.3.

After doing this, we need to define the IPSec VPN properties. We begin by defining the *transform set* that we want to use for these VPN connections. We will call this transform set *VPN-TRANSFORMS*.

```
Router1(config)#crypto ipsec transform-set VPN-TRANSFORMS ah-sha-hmac esp-sha-hmac
esp-3des
Router1(cfg-crypto-trans)#mode tunnel
```

The VPN will use the *esp-sha-hmac* and *esp-3des* transforms. This transform set is almost the same as the one in Recipe 12.3, but this time we have specified that this VPN should use tunnel mode with the *mode* command. In Recipe 12.3, IPSec was used to encrypt traffic in a GRE tunnel. However, here we are dealing with VPNs that terminate on a user workstation, so it is not possible to create a GRE tunnel before establishing the connection. This example uses tunnel mode, which is actually the default.

Because the workstation could, in principle, be anywhere on the Internet, we can't even define an IP address for it. But, to use IPSec on a Cisco router, we need to create a *crypto map*, which is a template for the SA that IPSec will use for this session. Fortunately, Cisco provides the ability to create dynamic crypto maps for precisely these types of situations:

```
Router1(config)#crypto dynamic-map VPN-USER-MAP 50
Router1(config-crypto-map)#description A dynamic crypto map for VPN users
Router1(config-crypto-map)#match address 115
```

```
Router1(config-crypto-map)#set transform-set VPN-TRANSFORMS
```

This creates a dynamic map called *VPN-USER-MAP*. The number (50) on the end of the line is a sequence number, similar to the sequence numbers used in route map statements. The router will look at all map entries in sequence until it finds a match. In this case, the match is decided by the *match address* command, which compares the IP addresses of packets to access-list 115. If the access list matches the addresses in the packet header, it will then apply the transform set that we created earlier.

The access list here blocks any packets whose destination addresses are either multicasts or local broadcasts. Obviously this type of traffic cannot possible be associated with a VPN:

```
Router1(config)#access-list 115 deny any 224.0.0.0 35.255.255.255
Router1(config)#access-list 115 deny any 172.25.1.255 0.0.0.0
Router1(config)#access-list 115 permit any any
```

In practice, you may want to use a more restrictive access list.

We can then build the actual crypto map that references this dynamic map. In the following command, we create a crypto map called, appropriately enough, `CRYPTOMAP`. This command is sequence number 10 in the definition of the map. In fact, it's the only command in the map's definition, but there could easily be others, including static crypto maps similar to the ones we discussed in Recipe 12.3 and Recipe 12.5. Usually you actually want to put any dynamic maps at the end of your crypto map. This is because dynamic maps work best as catchall conditions for unknown IP addresses. So, if there are any known IP addresses that require special attention, you need to configure them first, before the dynamic map statements.

You apply the crypto map to the interface that will be receiving the VPN requests:

```
Router1(config)#crypto map CRYPTOMAP 10 ipsec-isakmp dynamic VPN-USER-MAP
Router1(config)#interface FastEthernet0/1
Router1(config-if)#crypto map CRYPTOMAP
```

12.8.4 See Also

Recipe 12.3; Recipe 12.5; Chapter 4

Top

Recipe 12.9 Check IPSec Protocol Status

12.9.1 Problem

You want to check the status of a VPN.

12.9.2 Solution

There are several useful commands for displaying IPSec parameters.

The command *show crypto isakmp sa* shows all of the ISAKMP security associations:

```
Router1#show crypto isakmp sa
```

You can look at the IPSec security associations with this command:

```
Router1#show crypto ipsec sa
```

Even if you aren't using a key management protocol such as ISAKMP, you can see information on all of the active IPSec connections with the following command:

```
Router1#show crypto engine connections active
```

This closely related command will tell you about packet drops within the encryption engine:

```
Router1#show crypto engine connections dropped-packet
```

The *show crypto map* command gives information about all of the IPSec crypto maps that you have configured on your router, in use or not:

```
Router1#show crypto map
```

You can specify a particular crypto map with the *tag* keyword:

```
Router1#show crypto map tag TUNNELMAP
```

For information about dynamic crypto maps, you can use the following command:

```
Router1#show crypto dynamic-map
```

12.9.3 Discussion

The *show crypto isakmp sa* command lets you see information about the current state of any ISAKMP key exchanges that the router is involved in:

```
Router1#show crypto isakmp sa
dst             src             state          conn-id    slot
172.22.1.4      172.22.1.3      QM_IDLE              1       0

Router1#
```

Table 12-3 shows all of the possible ISAKMP SA states.

Table 12-3. ISAKMP SA states

Mode	State name	Description
Main Mode	MM_NO_STATE	There is an ISAKMP SA, but none of the parameters have been negotiated yet.
	MM_SA_SETUP	The devices have negotiated a set of parameters for the SA, but have not yet exchanged any key information.
	MM_KEY_EXCH	The devices have used the DH algorithm to create a common key, but they have not yet authenticated the session.
	MM_KEY_AUTH	The devices have authenticated the SA. They can now proceed to Quick Mode.
Aggressive Mode	AG_NO_STATE	There is an ISAKMP SA, but none of the parameters have been negotiated yet.
	AG_INIT_EXCH	The devices have initiated an Aggressive Mode exchange.
	AG_AUTH	The devices have completed an Aggressive Mode exchange and authenticated the SA. They can now proceed to Quick Mode.
Quick Mode	QM_IDLE	The SA is authenticated and ready for use.

We used Main Mode in all of the examples in this chapter. Aggressive Mode allows faster SA setup by combining SA parameter negotiation, key exchange, and authentication information into the same packet. This has the disadvantage of not hiding the identity information on the peer devices, however. In Main Mode exchanges, this identity information is exchanged separately in encrypted form. Main Mode is the default. Because the extra overhead is minimal, you generally don't need to resort to Aggressive Mode for ISAKMP.

Quick Mode is possible only after the initial ISAKMP exchange has happened at least once. The routers then use this mode when periodically renegotiating the SA information of an SA that has been

active for a while. Quick Mode can take advantage of the existing SA to encrypt its exchange.

You can use the following rather verbose command to look at IPSec SAs:

```
Router1#show crypto ipsec sa

interface:  FastEthernet0/1
     Crypto map tag: TUNNELMAP, local addr. 172.22.1.3

    local   ident (addr/mask/prot/port): (172.22.1.3/255.255.255.255/0/0)
    remote  ident (addr/mask/prot/port): (172.22.1.4/255.255.255.255/0/0)
   current_peer: 172.22.1.4
     PERMIT, flags={transport_parent,}
    #pkts encaps: 0, #pkts encrypt: 0, #pkts digest 0
    #pkts decaps: 0, #pkts decrypt: 0, #pkts verify 0
    #pkts compressed: 0, #pkts decompressed: 0
    #pkts not compressed: 0, #pkts compr. failed: 0, #pkts decompress failed: 0
   #send errors 0, #recv errors 0

     local crypto endpt.: 172.22.1.3, remote crypto endpt.: 172.22.1.4
    path mtu 1500, media mtu 1500
    current outbound spi: 0

    inbound esp sas:

    inbound ah sas:

    inbound pcp sas:

    outbound esp sas:

    outbound ah sas:

    outbound pcp sas:


  local   ident (addr/mask/prot/port): (172.22.1.3/255.255.255.255/47/0)
  remote  ident (addr/mask/prot/port): (172.22.1.4/255.255.255.255/47/0)
 current_peer: 172.22.1.4
   PERMIT, flags={origin_is_acl,transport_parent,parent_is_transport,}
  #pkts encaps: 466, #pkts encrypt: 466, #pkts digest 466
  #pkts decaps: 1156, #pkts decrypt: 1156, #pkts verify 1156
  #pkts compressed: 0, #pkts decompressed: 0
  #pkts not compressed: 0, #pkts compr. failed: 0, #pkts decompress failed: 0
 #send errors 1, #recv errors 0

     local crypto endpt.: 172.22.1.3, remote crypto endpt.: 172.22.1.4
    path mtu 1500, media mtu 1500
    current outbound spi: EB99FB6C

    inbound esp sas:
      spi: 0x5A48ACC4(1514712260)
```

```
    transform: esp-3des esp-sha-hmac ,
    in use settings ={Transport, }
    slot: 0, conn id: 2000, flow_id: 1, crypto map: TUNNELMAP
     sa timing: remaining key lifetime (k/sec): (4606612/3392)
    IV size: 8 bytes
    replay detection support: Y

inbound ah sas:

inbound pcp sas:

outbound esp sas:
  spi: 0xEB99FB6C(3952737132)
    transform: esp-3des esp-sha-hmac ,
    in use settings ={Transport, }
    slot: 0, conn id: 2001, flow_id: 2, crypto map: TUNNELMAP
     sa timing: remaining key lifetime (k/sec): (4607955/3392)
    IV size: 8 bytes
    replay detection support: Y

outbound ah sas:

outbound pcp sas:


Router1#
```

There is clearly a lot of information in this output. It breaks out the inbound and outbound information, and shows what crypto maps have been applied to which interfaces. It also includes information about the number of packets that the router has both been sent and received, as well as how much time remains before the SA must be renegotiated.

The *show crypto engine* commands allow you to see some of this same information in a more compact form. With the *connections active* keywords, this command tells you what interfaces are involved in IPSec SAs, the peer IP addresses, the algorithms used, and the number of packets sent and received through the encryption engine:

```
Router1#show crypto engine connections active

  ID Interface       IP-Address      State  Algorithm           Encrypt  Decrypt
   1 <none>          <none>          set    HMAC_SHA+3DES_56_C        0        0
2088 FastEthernet0/1 172.22.1.3      set    HMAC_SHA+3DES_56_C        0        5
2089 FastEthernet0/1 172.22.1.3      set    HMAC_SHA+3DES_56_C      202        0

Router1#
```

With the *connections dropped-packet* keywords, you get some simple statistics on dropped packets. In the following example, the encryption engine was forced to drop five packets because the router tried to send them before it had a valid connection:

```
Router1#show crypto engine connections dropped-packet

Packets dropped because of connection not established:
Interface             IP-Address           Drop Count
FastEthernet0/1       172.22.1.3                    5

Router1#
```

The command *show crypto map* displays information about all of the configured crypto maps on the router, including which interfaces are currently using them. Note that just because a particular interface is using a particular crypto map, it does not follow that there are any active IPSec SAs. It only means that you have applied this map to this interface using the *crypto map* interface configuration command:

```
Router1#show crypto map
        Interfaces using crypto map VPN-MAP:

Crypto Map "CRYPTOMAP" 10 ipsec-isakmp
        Dynamic map template tag: VPN-USER-MAP
        Interfaces using crypto map CRYPTOMAP:

Crypto Map "TUNNELMAP" 10 ipsec-isakmp
        Peer = 172.22.1.4
        Extended IP access list 116
            access-list 116 permit gre host 172.22.1.3 host 172.22.1.4
        Current peer: 172.22.1.4
         Security association lifetime: 4608000 kilobytes/3600 seconds
        PFS (Y/N): N
        Transform sets={ TUNNEL-TRANSFORM, }
        Interfaces using crypto map TUNNELMAP:
                FastEthernet0/1

Router1#
```

If you have several crypto maps configured on your router, you can look at a particular one with the*tag* keyword:

```
Router1#show crypto map tag TUNNELMAP
Crypto Map "TUNNELMAP" 10 ipsec-isakmp
        Peer = 172.22.1.4
        Extended IP access list 116
            access-list 116 permit gre host 172.22.1.3 host 172.22.1.4
        Current peer: 172.22.1.4
         Security association lifetime: 4608000 kilobytes/3600 seconds
        PFS (Y/N): N
        Transform sets={ TUNNEL-TRANSFORM, }
        Interfaces using crypto map TUNNELMAP:
                FastEthernet0/1

Router1#
```

If there are any dynamic maps, you can see more information about them with the following command:

```
Router1#show crypto dynamic-map
Crypto Map Template"VPN-USER-MAP" 50
        Extended IP access list 115
            access-list 115 permit tcp any port = 80 any
            access-list 115 permit tcp any any port = 80
            access-list 115 deny ip any 224.0.0.0 31.255.255.255
        Current peer: 0.0.0.0
         Security association lifetime: 4608000 kilobytes/3600 seconds
        PFS (Y/N): N
        Transform sets={ VPN-TRANSFORMS, }
Router1#
```

Top

Chapter 13. Dial Backup

Top

Introduction

Dial backup is an important feature in a reliable WAN design. If the primary link to a remote site fails, dial backup links can ensure that you don't lose all connectivity. Of course, the dial backup link will usually have significantly lower bandwidth than the primary link. However, the principle advantage of using a dialup connection for backup is that the link will only connect when required. The rest of the time the connection is down, which usually saves money, because you only pay for the access and avoid the connection charges.

The examples in this chapter are also useful for WAN designs in which the dial links are used as the primary connections. There are two common examples of networks like this. The first are networks that only connect when there is data to send. For example, in many retail environments, the remote store front sites only need to exchange data at the end of the day to update inventory and report the day's sales.

The other common type of network that uses only dialup connections involve sites that are in separate buildings, but within the same local dialing area. In this case, if the telephone company doesn't charge a usage fee, a pure dialup network can be a very cost-effective way of delivering low bandwidth WAN services.

Three technologies are commonly used for dialup links: standard analog telephone lines with asynchronous modems, switched 56Kbps synchronous digital service (sometimes called Centrex), and ISDN.

Analog Modems

Standard analog telephone lines with asynchronous modems are a reasonably effective dial backup technology, and they have the great advantage of being nearly ubiquitous: in regions where you can get no other network services, you can often get an analog telephone line. Further, most Cisco routers have an AUX port that supports an analog modem connection.

But this option has some important drawbacks. The first is that there are no guarantees about how much bandwidth you will get. Many analog modems are rated to speeds up to 56Kbps, but in practice you will rarely get this much throughput. It is more typical to see a practical bandwidth of between 9.6 and 44Kbps with asynchronous modems.

The second important problem with voice grade telephone lines is that they are susceptible to electrical noise, which can cause dropped packets and sometimes even dropped calls.

Switched 56Kbps Digital Service

Switched 56Kbps digital service, which also goes by the brand name Centrex in some areas, is a synchronous digital dialup technology. We recommend using this in regions that don't offer ISDN because it offers greater bandwidth and reliability than voice grade analog service. However, the number of local telephone companies that can offer switched 56Kbps but not ISDN is rapidly decreasing.

To use this technology, you need a synchronous serial port on your router, and an external Data Unit (DU), or synchronous modem.

ISDN

ISDN (Integrated Services Digital Network) is usually the best way to go for dialup networking. It has the highest bandwidth and the greatest reliability. And, when using ISDN with Cisco routers, you have the distinct advantage of being able to use built-in ISDN terminal adapters and Network Termination Type 1 (NT1) units, which reduces both the complexity and the costs of implementation and maintenance.

ISDN circuits come in two basic varieties called Basic Rate Interface (BRI) and Primary Rate Interface (PRI). A BRI circuit supports two 64Kbps B-channels and a 16Kbps D-channel that handles the signaling for the two B-channels. A PRI circuit, on the other hand, uses a single 64Kbps D-channel to support the signaling for 23 (if delivered through a T1 circuit) or 30 (for an E1 circuit) B-channels. Many network vendors will also sell PRI services on fraction T1 or E1 circuits, allowing smaller numbers of B-channels.

The D-channel is not usually used for user data, but Cisco routers allow you to bind the two B-channels together for a net 128Kbps link using the PPP multilink feature. Unlike analog modems, each of these channels operates at full-duplex, so you can send and receive simultaneously at the full channel speed.

It is possible to use the D-channel of a PRI circuit for user data, but only if the carrier has not configured this channel to manage the B-channels. In situations where you have multiple PRI circuits, it is possible to control all of the B-channels from the D-channel of the first PRI circuit, leaving the D-channels of the other circuits available for data. The advantages of doing this are slight, however.

Many organizations use BRI interfaces for remote branch devices, and PRI interfaces for central dialup circuits. This way you can save on physical ports by having many branches dial into a single central PRI circuit. By default, a PRI circuit can accept calls from remote ISDN circuits. ISDN circuits can also terminate calls from Centrex or switched 56Kbps type circuits without requiring any special hardware. Further, Cisco has analog modem cards for several routers such as the AS5x00 and 3600 series. These allow you to terminate analog calls from remote devices on the same PRI circuit. This is an extremely useful option because you can then configure all of your remote devices to dial to the same central ISDN PRI telephone number.

BRI interfaces come in two main varieties, called "S/T" and "U." Usually a BRI circuit is delivered and terminated on a U interface, which is a two-wire digital telephone line. The U interface connects to an NT1, which converts the U interface signaling to S/T interface signaling. The S/T interface then connects to a Terminal Adapter device, which allows you to connect the ISDN circuit to your equipment. Both S/T and U interfaces use standard RJ-45 cables.

Cisco allows you to eliminate some or all of these pieces of equipment, though, by offering a variety of ISDN hardware options. Many access routers come with an optional on-board Terminal Adapter, or can take an ISDN module with this functionality. The BRI interface is labeled "S/T" to indicate when the router has an on-board terminal adapter. You can connect this port to an external NT1 device, which in turn connects to the telephone company's circuit.

Cisco also has a variety of BRI modules that include an on-board NT1. These also use an RJ-45 connector, but they are labeled "U" to indicate that you should connect directly to the ISDN circuit. We generally prefer to implement ISDN on routers with on-board NT1 units because it simplifies implementation.

If you want to take full advantage of ISDN features, the router must at least have an on-board Terminal Adapter.

Estimating How Many Dialup Lines You Need

Many network engineers make the mistake of either under or overestimating how many dial backup lines they need to provide at their central site. In a hub-and-spoke WAN, you can easily estimate how many dialup lines you will need at the central site based on the probability failure for a branch's primary circuit.

The most common failure mode in any WAN is the so-called "last mile" failure, which means that the local loop circuit between the remote site and the WAN provider's Central Office (CO) breaks for some reason. The break could be due to a fiber cut, cross-connection problem, or (more common than anybody would like) human error. The provider will usually keep statistics on these problems, which they will use to define their Service Level Agreement (SLA) for each type of circuit.

The SLA effectively reflects a probability of a circuit failure. If, for example, your remote sites have a 99.9% SLA, this means that there is a 0.1% probability of failure. So, if you have a network with N circuits, each of which has the same probability of failure, P, you can use the following formula to calculate the probability of k simultaneous failures:

$$P(k,N) = N!\ P^{k}\ (1-P)^{(N-k)} / (k!\ (N-k)!)$$

The symbol "!" is a standard shorthand notation for the factorial function:

$$N! = N \times (N - 1) \times (N-2) \times \ldots \times 2 \times 1$$

So, for a WAN SLA of 99.9%, which is on the poor side (but typical), P is 0.1% (100% - 99.9%). If you have a hub-and-spoke WAN with N=100 circuits, the probability of there being a single circuit down is:

$$P(1,100) \sim 0.1 = 10\%$$

So roughly 10% of the time, you can expect to have one circuit down. Similarly, the probabilities of there being two or more simultaneous failures are given by:

$$P(2,100) \sim .5\%$$
$$P(3,100) \sim .02\%$$
$$P(4,100) \sim 0.00038\%$$
$$P(10,100) \sim 1.7 \times 10^{-15}\%$$

It's clear from this that the probability of 10 simultaneous failures is very small indeed. But just looking at probabilities can be deceptive because all of the numbers look small. We recommend multiplying these probabilities by the number of minutes in a year to get a better idea of how likely these failure scenarios actually are.

The probability of there being a single circuit failure is 10%, or 36.5 days per year. The probability of two simultaneous failures is 0.5%, which is roughly 44 hours per year. The probability of three simultaneous failure is .02%, or 105 minutes per year. And the probability of four simultaneous failures is .00038%, which is about two minutes per year.

So these are all things that you can expect to see happen at least once in the expected several year life span of this WAN. But the probability of 10 simultaneous failures is so small that you would expect it to happen roughly 5×10^{-10} seconds per year. Looking at this another way, if this failure condition lasted for one second, you would expect it to happen about once every billion years. Those are odds that most of us could live with.

By doing this sort of analysis, you can tell that having three dial backup circuits would probably come in handy at least once a year, and you might even need as many as four. But you're not likely to ever need 10.

However, it's important to bear in mind that this analysis assumes that these failures are not correlated. Depending on how your WAN provider implements your circuits, a single failure could affect several branches. So it is usually a good idea to apply a safety rule and double the number of circuits that this analysis suggests you will need. In this case, you probably need 4 circuits—but if you have 8 or 10, you should be more than safe.

Top

Recipe 13.1 Automating Dial Backup

13.1.1 Problem

You want automatic dial recovery in case a WAN link fails.

13.1.2 Solution

One of the most reliable ways of implementing dial backup on a Cisco router is to use a floating static default route, as follows:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#interface BRI0/0
Router1(config-if)#ip address 10.1.99.55 255.255.255.0
Router1(config-if)#encapsulation ppp
Router1(config-if)#dialer idle-timeout 300
Router1(config-if)#dialer map ip 10.1.99.1 name dialhost broadcast 95551212
Router1(config-if)#dialer load-threshold 50 either
Router1(config-if)#dialer-group 1
Router1(config-if)#isdn switch-type basic-ni
Router1(config-if)#isdn spid1 800555123400 5551234
Router1(config-if)#isdn spid2 800555123500 5551235
Router1(config-if)#ppp authentication chap
Router1(config-if)#ppp multilink
Router1(config-if)#exit
Router1(config)#username dialhost password dialpassword
Router1(config)#ip route 0.0.0.0 0.0.0.0  10.1.99.1 180
Router1(config)#dialer-list 1 protocol ip list 101
Router1(config)#access-list 101 deny eigrp any any
Router1(config)#access-list 101 permit ip any any
Router1(config)#router eigrp 55
Router1(config-router)#network 10.0.0.0
Router1(config-router)#end
Router1#
```

The matching configuration of the other end is shown in Recipe 13.2.

13.1.3 Discussion

This recipe includes several important features. First, note that we have configured dial backup using

an ISDN BRI interface on this router. So, we have to set up the ISDN configuration:

```
Router1(config)#interface BRI0/0
Router1(config-if)#isdn switch-type basic-ni
Router1(config-if)#isdn spid1 800555123400 5551234
Router1(config-if)#isdn spid2 800555123500 5551235
```

This site is connected to a National ISDN switch. So we have defined the switch type to be *basic-ni*. If this had been a PRI rather than a BRI, we would have used *primary-ni*. And, because it is a National ISDN switch, we also have to include the ISDN Service Profile Identifier (SPID) values. These define the telephone numbers associated with each of the two B-channels in the BRI. Note that the syntax includes essentially the same number twice:

```
Router1(config-if)#isdn spid1 800555123400 5551234
```

The first argument is the whole telephone number, including area code, with `00` tacked on the end. These extra two digits vary between different telephone companies. Sometimes this needs to be a different code such as `0101`. The telephone company can tell you the correct value to include.

The second number is not always required. This is essentially the phone number that you would need to call this B-channel from the other B-channel. In this example, the telephone company uses 7-digit local dialing, so we can eliminate the area code.

There are several different kinds of ISDN switches, and it's important to find out what your carrier uses to ensure that you configure the router properly.

For telephone companies that use AT&T switches:

```
Router1(config-if)#isdn switch-type basic-5ess
```

For telephone companies that use Nortel DMS100 switches:

```
Router1(config-if)#isdn switch-type basic-dms100
```

Telephone companies outside of North America often use different kinds of ISDN switches. In France you would use the following command:

```
Router1(config-if)#isdn switch-type vn3
```

In Australia, the telephone company uses TS013 ISDN switches:

```
Router1(config-if)#isdn switch-type basic-ts013
```

In Norway and New Zealand:

```
Router1(config-if)#isdn switch-type basic-net3
```

In Germany:

```
Router1(config-if)#isdn switch-type basic-1tr6
```

And, in Japan:

```
Router1(config-if)#isdn switch-type ntt
```

Please contact the local telephone company supplying the BRI circuit to ensure that you have the right switch type. And be sure to ask them whether you need to configure SPIDs on your router. Some switches require them, while others don't.

You can verify that you have your ISDN configuration working correctly with the*show isdn status* command:

```
Router1#show isdn status
Global ISDN Switchtype = basic-ni
ISDN BRI1/0 interface
     dsl 8, interface ISDN Switchtype = basic-ni
     Layer 1 Status:
     ACTIVE
     Layer 2 Status:
     TEI = 85, Ces = 1, SAPI = 0, State = MULTIPLE_FRAME_ESTABLISHED
     TEI = 86, Ces = 2, SAPI = 0, State = MULTIPLE_FRAME_ESTABLISHED
     TEI 85, ces = 1, state = 8(established)
         spid1 configured, spid1 sent, spid1 valid
     TEI 86, ces = 2, state = 8(established)
         spid2 configured, spid2 sent, spid2 valid
     Layer 3 Status:
     0 Active Layer 3 Call(s)
     Activated dsl 8 CCBs = 0
     The Free Channel Mask:  0x80000003
Total Allocated ISDN CCBs = 2
Router1#
```

In this case, you can see that you have an "active" status at Layer 1, and both of the Terminal Endpoint Identifiers (TEI) are in a "MULTIPLE_FRAME_ESTABLISHED" state. This means that the router is talking with the telephone company's ISDN switch and both of the B-channels are ready to go. This display also says that there are currently no active calls at Layer 3. As an aside, we should point out that this refers to the ISDN circuit's Layer 3, and not the IP network layer. When the router places a call, it will establish a PPP connection, which will support IP.

The actual dialing is done by the *dialer map* command:

```
Router1(config)#dialer-list 1 protocol ip list 101
Router1(config)#access-list 101 deny eigrp any any
Router1(config)#access-list 101 permit ip any any
Router1(config)#interface BRI0/0
Router1(config-if)#dialer map ip 10.1.99.1 name dialhost broadcast 95551212
Router1(config-if)#dialer-group 1
```

In this case, the dialer map says that, to reach the IP address `10.1.99.1`, which is a router called dialhost, it should dial the phone number `95551212`. Note that we have included a "9" at the start of this phone number. Again, you will need to ask your local telephone company whether there is a special code digit. We have seen places where we needed a 9, an 8, or nothing at all.

The *broadcast* keyword in this command allows both multicast and broadcast traffic to use this dialup link. This is extremely important for routing protocols such as EIGRP, RIPv2, and OSPF, which use multicasts for sending their updates between routers. This example uses EIGRP, so we need to include this keyword.

With this type of dialer configuration, you also need to define a dialer group. In this case, we have assigned this interface to dialer group number 1. You configure the behavior of this dialer group with the *dialer-list* statement, which defines what an interesting packet is for this network.

An interesting packet is one that will bring up the dialer, or keep it active if it is already up. If the circuit is up, the router will reset the idle timer every time it sees an interesting packet. The result is that as long as there are interesting packets to send, the router will keep the dial session active. Otherwise, it will disconnect the call when the idle timer expires. This is particularly important when you are calling long- distance numbers. If the wrong packets are considered interesting, it could mean long call and an expensive phone bill.

So we have associated the dialer list with an access list that specifies what is interesting. In this case, all IP packets except EIGRP are interesting. It's important to remember that EIGRP packets will still pass through the dial link normally. But if the link is not active, an EIGRP packet is not sufficient to bring it up. And, if the link is active, the presence of EIGRP packets alone won't prevent the router from dropping it.

This is extremely important, because, as we discussed in Chapter 7, the router will send an EIGRP HELLO packet every few seconds by default. But we don't want the link to remain active unless there is real user traffic to send. If you are using a different routing protocol, you should specify its update packets here instead.

Sometimes you do want the link to remain active all the time. For example, the administrators of some small WANs like to keep ISDN sessions nailed up all the time (usually because they only pay an access charge, and not a usage or long-distance charge). So, if the session drops for any reason, they want it to immediately dial up again. In this case, you could replace the access list with a new one that finds all traffic interesting:

```
Router1(config)#access-list 101 permit ip any any
```

It's easier still if you modify the *dialer-list* command to make all IP traffic interesting:

```
Router1(config)#dialer-list 1 protocol ip permit
```

When the router dials, it will use Point-to-Point Protocol (PPP) to carry Layer 3 protocols such as IP. So

you need to define several PPP parameters:

```
Router1(config)#interface BRI0/0
Router1(config-if)#encapsulation ppp
Router1(config-if)#ppp authentication chap
Router1(config-if)#exit
Router1(config)#username dialhost password dialpassword
```

The *encapsulation* command simply tells the router to use PPP as its Layer 2 protocol. But, because you don't want just anybody dialing into this dialhost router, it's a good idea to include some authentication. In this case we have configured the router to use Challenge Handshake Authentication Protocol (CHAP) for authenticating PPP sessions. This basically means that both this router and the router it dials to will exchange usernames and passwords when they connect. The username for this router is the router's name. We define the username and password for the other router with the *username* command.

Cisco also supports PPP authentication scheme called Password Authentication Protocol (PAP). CHAP is much more secure because it only passes passwords in encrypted form rather than clear-text, as PAP does. CHAP is no more complex to set up, and presents no appreciable extra load on the router's resources. So we strongly recommend using CHAP rather than PAP.

Because this is an ISDN BRI interface, we would like to be able to use both of the B-channels to increase the available bandwidth:

```
Router1(config)#interface BRI0/0
Router1(config-if)#dialer load-threshold 50 either
Router1(config-if)#ppp multilink
```

The command *ppp multilink* means that this PPP session can be split across several physical connections. This feature allows full load balancing and packet sequencing across all of the connections in the multilink bundle. In this case, we want to bond the two ISDN B-channels into a single 128Kbps PPP link. By default, the router will only use one of these channels, whichever one is available. The *dialer load-threshold* command specifies the rule that the router will use to bring up the second link. In this case, we have specified that, if the traffic utilization in either direction (input or output) reaches 50%, then the router should bring up the second channel.

We have also modified the default idle timeout:

```
Router1(config)#interface BRI0/0
Router1(config-if)#dialer idle-timeout 300
```

By default, the router will drop the dial session if there have been no interesting packets for 120 seconds. We have increased this value to 300 seconds. Because ISDN dials so quickly, this is not vital. But with asynchronous modem dialup, it can take up to a full minute to establish a new session. You often need to increase the idle timer is to make sure that the primary connection is up and stable before disconnecting the backup circuit. It is a good idea to wait for the routing protocol to converge, and to ensure that the primary circuit isn't simply bouncing up and down. You also have to trade off between

the time required to establish a new session and the cost of any long-distance charges on this line. We generally recommend using an idle timeout period of five minutes, as shown in the example.

Finally, we come to one of the most important features of this configuration, the trigger condition. This router will dial whenever it has traffic to send to the IP address `10.1.99.1`, which is the IP address of the dialhost router itself. User traffic will be directed to end devices such as servers, not to routers. The only way to bring up this dial interface is if this router needs to send an interesting packet to the dial router's IP address. This is where the floating static route comes in.

In Chapter 5 we discussed floating static routes. These are routes whose administrative distances are so high that any dynamically learned route to the same destination will be better. The router will only install this static route if the dynamic routing protocol can't offer anything better:

```
Router1(config)#ip route 0.0.0.0 0.0.0.0  10.1.99.1 180
```

In this particular case, the routing protocol is EIGRP, which has default administrative distance of 90 for all internal routes and 170 for external routes. So, by creating this static default route with a metric of 180, we ensure that the router will never use it if it has anything better.

The net result is that, if the primary link fails, EIGRP will lose all of its routes. So the router will install the floating static route to handle any user data packets that it needs to transmit. Since this route points to the far end of the dial link, the router must bring up the dial connection.

The nice thing about this way of triggering dial backup is that it is extremely robust. Anything that causes you to lose connectivity for any reason will trigger the dial backup. This is better than the backup interface solution described in Recipe 13.4, for example, because it doesn't require loss of physical connectivity to trigger the backup.

Also, as we discuss in Recipe 13.4 (which uses the backup interface method for triggering dial backup), the floating static configuration allows you to have the interface remain up but not connected when the primary circuit is working. In the case of ISDN, this means that you can use the *show isdn* commands that we discuss in Recipe 13.7 to ensure that your circuit is still working.

One of the most useful features of this type of trigger mechanism is that you can test the dial backup easily. If you look at the dialer list, you will see that all the router needs to initiate a dial session is to have a packet to send to the far end that matches the dialer list. So, in this particular example, you could easily bring up a dial session for testing by just logging into the remote router and pinging the IP address of the dial backup router:

```
Router1#ping 10.1.99.1
```

13.1.4 See Also

Recipe 13.2; Recipe 13.4; Chapter 5; Chapter 7

Top

Recipe 13.2 Using Dialer Interfaces

13.2.1 Problem

You want to treat several physical interfaces as a single dialer.

13.2.2 Solution

If you have several physical interfaces on your router that you want to treat as a single dialer, particularly for PPP multilink channel bonding, you can create a logical dialer interface:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#interface BRI0/0
Router1(config-if)#encapsulation ppp
Router1(config-if)#dialer pool-member 1
Router1(config-if)#isdn switch-type basic-ni
Router1(config-if)#isdn spid1 800555123400 5551234
Router1(config-if)#isdn spid2 800555123500 5551235
Router1(config-if)#ppp authentication chap
Router1(config-if)#exit
Router1(config)#interface BRI0/1
Router1(config-if)#encapsulation ppp
Router1(config-if)#dialer pool-member 1
Router1(config-if)#isdn switch-type basic-ni
Router1(config-if)#isdn spid1 800555123600 5551236
Router1(config-if)#isdn spid2 800555123700 5551237
Router1(config-if)#ppp authentication chap
Router1(config-if)#exit
Router1(config)#interface Dialer1
Router1(config-if)#ip address 10.1.99.55 255.255.255.0
Router1(config-if)#encapsulation ppp
Router1(config-if)#dialer remote-name dialhost
Router1(config-if)#dialer pool 1
Router1(config-if)#dialer idle-timeout 300
Router1(config-if)#dialer string 95551212
Router1(config-if)#dialer load-threshold 50 either
Router1(config-if)#dialer-group 1
Router1(config-if)#ppp authentication chap
Router1(config-if)#ppp multilink
Router1(config-if)#exit
Router1(config)#username dialhost password dialpassword
Router1(config)#ip route 0.0.0.0 0.0.0.0  10.1.99.1 180
```

```
Router1(config)#dialer-list 1 protocol ip list 101
Router1(config)#access-list 101 deny eigrp any any
Router1(config)#access-list 101 permit ip any any
Router1(config)#router eigrp 55
Router1(config-router)#network 10.0.0.0
Router1(config-router)#end
Router1#
```

Dialer interfaces are particularly useful for the server side, where you can use them to bond together several ISDN BRI or PRI circuits:

```
dialhost#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
dialhost(config)#username Router1 password dialpassword
dialhost(config)#controller T1
dialhost(config-controller)#framing esf
dialhost(config-controller)#clock source line primary
dialhost(config-controller)#linecode b8zs
dialhost(config-controller)#pri-group timeslots 1-24
dialhost(config-controller)#exit
dialhost(config)#interface Serial0:23
dialhost(config-if)#encapsulation ppp
dialhost(config-if)#dialer rotary-group 1
dialhost(config-if)#dialer-group 1
dialhost(config-if)#isdn switch-type primary-dms100
dialhost(config-if)#isdn not-end-to-end 56
dialhost(config-if)#exit
dialhost(config)#interface Dialer1
dialhost(config-if)#ip address 10.1.99.1 255.255.255.0
dialhost(config-if)#encapsulation ppp
dialhost(config-if)#dialer in-band
dialhost(config-if)#dialer idle-timeout 300
dialhost(config-if)#dialer-group 1
dialhost(config-if)#no peer default ip address
dialhost(config-if)#ppp authentication chap
dialhost(config-if)#ppp multilink
dialhost(config-if)#exit
dialhost(config)#access-list 101 deny eigrp any any
dialhost(config)#access-list 101 permit ip any any
dialhost(config)#dialer-list 1 protocol ip list 101
dialhost(config)#router eigrp 55
dialhost(config-router)#network 10.0.0.0
dialhost(config-router)#end
dialhost#
```

13.2.3 Discussion

This example is similar to Recipe 13.1, but this time we have created a logical *`Dialer1`* interface instead of using a *dialer map* command. The effect is the same, but with dialer interfaces you have the advantage of being able to bond together several different physical links into a single PPP multilink

bundle.

In the first example, we have included two ISDN BRI interfaces, giving us an effective total bandwidth of 256Kbps for the backup link. However, as inRecipe 13.1, we have included a *dialer load-threshold* command so the router will bring up these additional B channels only if it requires them.

There are a couple of important differences between the first example in this recipe and the one in Recipe 13.1. First, notice that we have not included any IP addresses or dialer configuration information on the physical interfaces. Instead, we put all of this information in the configuration of the logical dialer interface.

Then, to associate these physical interfaces with this particular logical interface, we use the *dialer pool-member* command on the physical interfaces and the *dialer pool* command on the dialer interface. In this example, we have created dialer pool number 1 on the Dialer1 interface and assigned the two BRI interfaces to this pool. The dialer interface number is arbitrary. The only thing that matters is that the dialer pool numbers match the dialer pool member numbers.

Because there is no dialer map command to define the telephone number to call, the destination hostname, and the destination IP address, we have to configure these separately. First we set up the remote hostname and the dialer string (which defines the destination phone number) as follows:

```
Router1(config)#interface Dialer1
Router1(config-if)#dialer remote-name dialhost
Router1(config-if)#dialer string 95551212
```

And, as in Recipe 13.1, we include a floating static route to trigger the dial backup:

```
Router1(config)#ip route 0.0.0.0 0.0.0.0  10.1.99.1 180
```

The rest of the configuration is essentially the same as in Recipe 13.1.

The second example in this recipe shows a sample server side configuration. In many ways it is similar to the branch, but there are also a few key differences. The first difference is that the server is configured to use a PRI rather than a BRI circuit. In this case, the router uses a built-in T1 CSU, so we need to define the framing, line coding, and how the T1 time slices work:

```
dialhost(config)#controller T1
dialhost(config-controller)#framing esf
dialhost(config-controller)#clock source line primary
dialhost(config-controller)#linecode b8zs
dialhost(config-controller)#pri-group timeslots 1-24
dialhost(config-controller)#exit
```

This represents the most common options, Extended Super Frame (ESF) framing with Binary 8-Zero Substitution (B8ZS) line coding. And we will draw the clock from the circuit, rather than generating it in the router. The most important part of this is the definition of the T1 time slots. In this case, we have grouped all 23 B-channels and the D-channel into a single PRI group. This reflects the fact that we

purchased this circuit as a whole T1. However, you could just as easily work with a fractional T1 PRI circuit that includes only some of the available time slots. Please see Chapter 16 for more information on the *controller* command.

Once we have defined the T1 time slots for the PRI circuit, we can configure the circuit for dialup:

```
dialhost(config)#interface Serial0:23
dialhost(config-if)#encapsulation ppp
dialhost(config-if)#dialer rotary-group 1
dialhost(config-if)#dialer-group 1
dialhost(config-if)#isdn switch-type primary-dms100
dialhost(config-if)#isdn not-end-to-end 56
dialhost(config-if)#exit
```

The name of this interface, `Serial0:23`, means that we are working with the circuit attached to interface `Serial0`, and that it includes 23 time slices. In this example, the telephone company's ISDN switch is a Nortel DMS100, so we have to configure this with the *isdn switch-type* command. The *encapsulation ppp* and the *dialer-group* commands are familiar from previous examples, but there are a couple of other options here.

The first new feature is the *dialer rotary-group* command. This is a useful variation on some of the dialer commands that we discussed earlier. Because the argument of this command is the number 1, this physical interface is assigned to a rotary group that is associated with the virtual interface, `Dialer1`. A rotary group is similar to any other dialer group, but it allows multiple simultaneous connections to different remote routers. This wasn't necessary for the branch routers, because they dial only to the one central router. But the host router must be able to accept calls from many branches at once.

The primary router doesn't require dialer map statements to accept inbound calls. These are only necessary for outbound calls. When the router receives a new inbound connection, it will create a dynamic map to associate the IP address with the dial connection.

The last command in this configuration is often required when using ISDN calls between different telephone companies, particularly for long-distance calls:

```
dialhost(config-if)#isdn not-end-to-end 56
```

By default, the router will assume that all calls use 64Kbps ISDN B-channels. But some regions use 56Kbps instead of 64. And, worse still, sometimes you have a long distance call that starts and ends at 64Kbps but has a hidden leg of 56Kbps in the middle of the carrier's network. In all of these cases, the router will drop the call by default, because of the speed mismatch. This command manually forces the router to use 56Kbps for all calls to prevent these speed mismatch problems.

13.2.4 See Also

Recipe 13.1; Chapter 16

Top

Recipe 13.3 Using an Async Modem on the AUX Port

13.3.1 Problem

You want to connect a standard asynchronous modem to the router's AUX port and use it for dial backup.

13.3.2 Solution

Many Cisco routers include an AUX port that is a low-speed asynchronous serial interface that can connect to a standard modem and support PPP:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router2(config)#interface Async65
Router2(config-if)#encapsulation ppp
Router2(config-if)#dialer in-band
Router2(config-if)#dialer pool-member 1
Router2(config-if)#ppp authentication chap
Router2(config-if)#async default routing
Router2(config-if)#exit
Router2(config)#interface Dialer1
Router2(config-if)#ip address 10.1.99.56 255.255.255.0
Router2(config-if)#encapsulation ppp
Router2(config-if)#dialer remote-name dialhost
Router2(config-if)#dialer pool 1
Router2(config-if)#dialer idle-timeout 300
Router2(config-if)#dialer string 95551212
Router2(config-if)#dialer-group 1
Router2(config-if)#ppp authentication chap
Router2(config-if)#exit
Router2(config)#line aux 0
Router2(config-line)#modem inout
Router2(config-line)#transport input all
Router2(config-line)#no exec
Router2(config-line)#speed 115200
Router2(config-line)#exit
Router2(config)#username dialhost password dialpassword
Router2(config)#ip route 0.0.0.0 0.0.0.0  10.1.99.1 180
Router2(config)#dialer-list 1 protocol ip list 101
Router2(config)#access-list 101 deny eigrp any any
Router2(config)#access-list 101 permit ip any any
Router2(config)#router eigrp 55
```

```
Router2(config-router)#network 10.0.0.0
Router2(config-router)#end
Router2#
```

13.3.3 Discussion

Much of this configuration is similar to the ISDN configuration shown in Recipe 13.2. It uses a dialer interface in exactly the same way. But here, because there is only one async modem in this example, we can't benefit from PPP multilink.

The first part of this configuration example sets up the AUX port to run PPP and associates it with a dialer pool:

```
Router2(config)#interface Async65
Router2(config-if)#encapsulation ppp
Router2(config-if)#dialer in-band
Router2(config-if)#dialer pool-member 1
Router2(config-if)#ppp authentication chap
Router2(config-if)#async default routing
```

The only thing here that hasn't appeared in a previous example is the *async default routing* command. This command allows the async interface to support a routing protocol such as EIGRP. By default, routing protocols are disabled on async interfaces, so you need to enable it.

The number of this particular interface, `Async65`, wasn't selected at random. The router automatically assigns a line number to every interface that can be used for terminal access (including VTY lines, AUX lines, and console lines), and the line number varies from router to router, depending on the hardware configuration. So we used the *show lines* command to see which line number corresponded to the AUX port on this router:

```
Router1#show lines
   Tty Typ     Tx/Rx    A Modem  Roty AccO AccI   Uses   Noise  Overruns   Int
     0 CTY              -    -      -    -    -      0       0     0/0       -
    65 AUX   9600/9600  -    -      -    -    -      0       0     0/0       -
*   66 VTY              -    -      -    -    -     10       0     0/0       -
*   67 VTY              -    -      -    -    -     19       0     0/0       -
    68 VTY              -    -      -    -    -      3       0     0/0       -
    69 VTY              -    -      -    -    -      0       0     0/0       -
    70 VTY              -    -      -    -    -      0       0     0/0       -
    71 VTY              -    -      -    -    -      0       0     0/0       -
    72 VTY              -    -      -    -    -      0       0     0/0       -
    73 VTY              -    -      -    -    -      0       0     0/0       -
    74 VTY              -    -      -    -    -      0       0     0/0       -
    75 VTY              -    -      -    -    -      0       0     0/0       -

Line(s) not in async mode -or- with no hardware support:
1-64

Router1#
```

As you can see, the AUX port is on line 65 on this router. It's important to do this before you attempt any of the rest of the configuration, so you know what to configure.

When you use the AUX port for dial backup, you also need to configure the terminal line information for this physical port:

```
Router2(config)#line aux 0
Router2(config-line)#modem inout
Router2(config-line)#transport input all
Router2(config-line)#no exec
Router2(config-line)#speed 115200
```

The first command here is *modem inout*, which configures the router to allow access to the modem, and vice versa. Then we added the command *transport input all* so that there are no restrictions on the protocols that may use this port.

The *no exec* command is extremely important when using async dial, and almost universally ignored in Cisco references. By default, the router will start an EXEC session on your AUX port. So, if you plug a terminal into this port, you will get a login prompt. Unfortunately, your modem doesn't know what to do with a login prompt. At best, it will just ignore it, so disabling the EXEC session is simply good form. But, at worst, we have seen problems where the modem attempts to respond to the login prompt, the EXEC session interprets this as a bad login attempt and puts up a new prompt, and the modem again attempts to respond. The result can be high CPU utilization and, more importantly, this activity will prevent the router from dialing. We strongly recommend disabling the EXEC session on any async dial ports, as we have done here.

The last command in this section sets the line speed. It's important to remember that this is the speed between the router and the modem. The actual dial session will have a much lower net speed, likely less than 56Kbps. However, it's a good idea to make the line speed as fast as the modem can support. This will ensure that you get the best possible speed. Note that the default speed here is only 9.6Kbps—if you don't increase this value, you will not be able to get the full advantage of the compression capabilities of modern modems.

13.3.4 See Also

Recipe 13.1; Recipe 13.2

Top

Recipe 13.4 Using Backup Interfaces

13.4.1 Problem

You want to configure a router to dial only if it sees a physical failure on the primary WAN interface.

13.4.2 Solution

Cisco routers can watch the physical signals on an interface and trigger a backup interface if the primary fails. The router will automatically drop the call after the primary circuit comes back up:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#interface Serial0/0
Router1(config-if)#backup delay 0 300
Router1(config-if)#backup interface BRI0/0
Router1(config-if)#encapsulation frame-relay
Router1(config-if)#down-when-looped
Router1(config-if)#exit
Router1(config)#interface Serial0/0.1 point-to-point
Router1(config-subif)#ip address 10.1.1.10 255.255.255.252
Router1(config-subif)#frame-relay interface-dlci 50
Router1(config-subif)#exit
Router1(config)#interface BRI0/0
Router1(config-if)#ip address 10.1.99.55 255.255.255.0
Router1(config-if)#encapsulation ppp
Router1(config-if)#dialer idle-timeout 300
Router1(config-if)#dialer map ip 10.1.99.1 name dialhost broadcast 95551212
Router1(config-if)#dialer load-threshold 50 either
Router1(config-if)#dialer-group 1
Router1(config-if)#isdn switch-type basic-ni
Router1(config-if)#isdn spid1 800555123400 5551234
Router1(config-if)#isdn spid2 800555123500 5551235
Router1(config-if)#ppp authentication chap
Router1(config-if)#ppp multilink
Router1(config-if)#exit
Router1(config)#dialer-list 1 protocol ip permit
Router1(config)#end
Router1#
```

13.4.3 Discussion

In this example, the primary WAN interface is a Frame Relay connection. Please see Chapter 10 for more information about Frame Relay configuration. However, this would work as well on just about any kind of interface. The main reason we used Frame Relay is to show that you have to put the backup commands on the physical interface, not on any subinterfaces or virtual interfaces. If this router loses physical signaling on the serial interface, it will automatically bring up the dial backup. The key to this configuration method is the *backup* command, which you associate with the primary interface:

```
Router1(config)#interface Serial0/0
Router1(config-if)#backup delay 0 300
Router1(config-if)#backup interface BRI0/0
```

In this case you can see that the backup interface for this serial port is the ISDN interface, `BRI0/0`. We also included a *backup delay* command, which specifies two times. The first parameter tells the router how long it should wait before bringing up the backup after it loses signals on this primary interface. In this case, we don't want to wait. If there is a failure, we want the backup to activate immediately. However, in some cases, you might want to delay slightly to save money on backup charges in case the primary comes back again right away. So, if you wanted to wait 15 seconds before dialing, you could configure it like this:

```
Router1(config-if)#backup delay 15 300
```

The second number tells the router how long to wait after the primary recovers before dropping the dial connection. If you're using Frame Relay, it can take a minute or more after you see physical signals before there is end-to-end connectivity. So it is important to keep the backup link active until everything has stabilized. Also, a link will sometimes bounce up and down if there are electrical problems. Specifying a sensible delay before dropping the backup link ensures helps with link stability.

We have also included the *down-when-looped* command on the primary interface:

```
Router1(config)#interface Serial0/0
Router1(config-if)#down-when-looped
```

The dial backup will trigger only if this interface line protocol is in a "down" state. Normally, when you put a circuit into a loopback state for testing, the router considers the interface to be in an "up" state, but looped. However, when it's in this diagnostic state, the circuit will not pass any data. So, by configuring *down-when-looped*, we ensure that the backup will trigger if somebody runs a loopback test (perhaps unintentionally) on the primary circuit.

In general, we don't recommend using the backup interface method for dial backup. There are many types of WAN problems in which you will lose connectivity, but you don't lose physical signaling on the interface. For example, in the Frame Relay case again, there could be a problem in the cloud that causes you to lose your virtual circuit. Or you might be connected to a faulty network termination device that keeps signals active even though it doesn't have a real connection. The floating static method given in Recipe 13.1 and Recipe 13.2 is much more robust than the backup interface method.

There is another important disadvantage to using the backup interface method. The router will keep backup interfaces disabled until it needs to dial. This causes two problems.

First, it means that you have to wait longer to dial because the router has to first establish physical connectivity with the backup network. In the case of ISDN, this can take 10-15 seconds.

The second problem is that, with ISDN interfaces, you lose the ability to see the state of the ISDN connection. Normally, if an ISDN interface is connected but not dialed, you can use the *show isdn status* command to verify that it is talking to the carrier's switch correctly, as we discussed in Recipe 13.1. However, since the backup interface is disabled with the method shown in the current recipe, you can't easily verify that your backup circuit is working without failing the primary circuit.

There is actually an interesting way to get around this last problem, though. Instead of using a physical interface (such as an ISDN port as we did in this example), you could make the backup interface be a dialer interface, as we discussed in Recipe 13.2. In this case, the dialer interface will remain down when the primary is working, but the ISDN interface will still be up. This means that you will be able to use the various show isdn commands as you can with the other methods.

We do not recommend using the backup interface method for dial backup, but there is one interesting extra option to the backup interface configuration that can be useful in some situations. In addition to triggering the backup circuit when the primary circuit fails, you can configure the router to trigger the backup circuit when the load on the primary circuit gets heavy. This is a form of bandwidth on demand:

```
Router1(config)#interface Serial0/0
Router1(config-if)#backup load 75 25
```

This command triggers the dial backup when the load on the primary interface rises to about 75%, and deactivates it when the load drops below 25%. Note, however, that to be really useful as additional bandwidth, you have to make sure that the routing over this new connection makes sense. In particular, it doesn't help much unless the routing protocol sees the two paths as equal and shares the load between them. This will generally require some careful metric tuning in your routing protocol, and it will almost certainly require that the dial backup circuit terminates on the same router as the primary circuit. Otherwise two-way load sharing will be very difficult to arrange.

13.4.4 See Also

Recipe 13.1; Chapter 10

Top

Recipe 13.5 Using Dialer Watch

13.5.1 Problem

You want to use Cisco's dialer watch feature to trigger dial backup.

13.5.2 Solution

The dialer watch feature allows the router to track a particular destination IP address in its routing table. If all of the tracked IP addresses disappear from the routing table, the router automatically triggers the dial backup connection:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#interface BRI0/0
Router1(config-if)#ip address 10.1.99.55 255.255.255.0
Router1(config-if)#encapsulation ppp
Router1(config-if)#dialer map ip 10.1.1.0 name dialhost broadcast 95551212
Router1(config-if)#dialer map ip 10.2.0.0 name dialhost broadcast 95551212
Router1(config-if)#dialer map ip 10.1.99.1 name dialhost broadcast 95551212
Router1(config-if)#dialer load-threshold 50 either
Router1(config-if)#dialer-group 1
Router1(config-if)#dialer watch-group 1
Router1(config-if)#dialer watch-disable 300
Router1(config-if)#isdn switch-type basic-ni
Router1(config-if)#isdn spid1 800555123400 5551234
Router1(config-if)#isdn spid2 800555123500 5551235
Router1(config-if)#ppp authentication chap
Router1(config-if)#ppp multilink
Router1(config-if)#exit
Router1(config)#username dialhost password dialpassword
Router1(config)#dialer-list 1 protocol ip list 101
Router1(config)#access-list 101 deny eigrp any any
Router1(config)#access-list 101 permit ip any any
Router1(config)#router eigrp 55
Router1(config-router)#network 10.0.0.0
Router1(config-router)#exit
Router1(config)#dialer watch-list 1 ip 10.1.1.0 255.255.255.0
Router1(config)#dialer watch-list 1 ip 10.2.0.0 255.255.0.0
Router1#
```

13.5.3 Discussion

This configuration is similar to that of Recipe 13.1, but this time the router uses the dialer watch feature to trigger the dial backup. The *dialer watch-group* command configures the backup interface to belong to a particular group. Usually you would configure only one such group on a router, but there is nothing to prevent you from having several different watch groups, each with different dial interfaces.

The watch group in this example looks for two prefixes in the routing tables, `10.1.1.0/24` and `10.2.0.0/16`. If both of these routes drop out of the routing table, the router will automatically bring up the dial interface. Note that all of the watched routes must disappear before the router will dial.

We have configured several dialer map statements for this example. The same dialer map statement that we used in Recipe 13.1 defines the basic IP routing of the interface. We have also included two watch lists, one for each of the watched IP addresses.

We used the same dialer list configuration here as we did in Recipe 13.1. This is because we still don't want routine EIGRP packets to bring up the dial interface.

After it triggers the dial backup, the dialer watch configuration will keep track of the primary interface by periodically looking for the watched IP addresses in its routing table. The dialer watch feature will consider the primary circuit active only if the watched routes exist in the routing table, and do not point through the dial interface. We have configured a delay of 300 seconds before the router drops the dial backup after deciding (based on the routing tables) that the primary interface has recovered:

```
Router1(config-if)#dialer watch-disable 300
```

In general, we still prefer to use floating static routes to trigger dial backup, as in Recipes Recipe 13.1 and Recipe 13.2. This is because they offer greater control and flexibility.

13.5.4 See Also

Recipe 13.1; Recipe 13.2

Top

Recipe 13.6 Ensuring Proper Disconnection

13.6.1 Problem

You want to ensure that the dial backup line disconnects properly when the primary link recovers.

13.6.2 Solution

Sometimes funny things happen when the primary link comes back and the backup link has not yet disconnected. These problems are usually due to poor routing metrics, which can cause at least one of the routers to prefer the dial path, even if the primary is available. The easiest way to handle these problems is to use bandwidth commands to ensure that the primary is the better path:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#interface Serial0/0.1 point-to-point
Router1(config-subif)#bandwidth 56
Router1(config-subif)#exit
Router1(config)#interface BRI0/0
Router1(config-subif)#bandwidth 54
Router1(config-subif)#end
Router1#
```

13.6.3 Discussion

This example assumes that you have a Frame Relay connection using a 56Kbps primary link, and an ISDN dial backup connection. The problem is that the default ISDN interface bandwidth is 64Kbps. So, if both the primary and the backup are active at the same time, the routing protocol will see the backup link as the preferred path. As a result, there will always be interesting traffic flowing through the backup link, and the router will fail to disconnect the dialup session properly.

To address this problem, we have configured a bogus bandwidth on the BRI interface. This will ensure that if both primary and backup are active simultaneously, the primary will be the better path. The only stipulation is that the value must be lower for the backup path.

Actually, this change affects only the routing decisions for traffic going from this router to the dial router. It is not difficult to get a situation where traffic from the branch to the hub site takes the primary path, while outbound packets to the branch take the backup path. So you have to be careful to adjust the values on both ends.

Dial backup situations can get more complicated in some networks. Perhaps the most insidious problems happen when the router responsible for the primary WAN link summarizes a large number of branch IP address ranges. This summarization is normally a good thing because it simplifies the global routing tables and improves overall performance.

However, suppose the branch has a LAN segment that uses `10.11.100.0/24`. And suppose the primary WAN router connects to many branches, offering the network core only a summary route that describes all of the branches as `10.11.0.0/16`. When the primary WAN link for this branch breaks, the branch router dials to a dedicated dial backup router. The dynamic routing works because, although the primary WAN router advertises a summary route that includes this branch's LAN address, the backup router advertises a more specific route. So other devices in the network core can route to the branch using the more specific route from the backup router.

The problem appears when the primary link recovers. There are now two paths available. By adjusting bandwidths as shown in this recipe, the branch router knows that it should switch to using the primary link. But the rest of the network doesn't see any change. The primary router continues to present only a summary route for all of its branches, and the dial backup host still presents the more specific route for this particular branch. So all outbound traffic to the branch still uses the backup link.

We know of no simple solution to this problem, except to ensure that the branch router is responsible for dropping the dial connection with an appropriate timeout. The alternatives are to avoid summarization or to have the dial backup router be the same physical device as the primary WAN router. You can help the branch router to switch to the primary, though, by ensuring that the dial host router sends only a default route, rather than a full routing table. This way, when the primary circuit recovers, the branch router will use the more specific routes that it learns through the primary circuit, thus quickly removing any interesting packets from the backup link. Then the branch router can take the responsibility for dropping the dial connection.

Top

Recipe 13.7 View Dial Backup Status

13.7.1 Problem

You want to check on the dialer status of a router.

13.7.2 Solution

Here are some useful commands for looking at the status of a dial backup link. For dial backup that uses the floating static or dialer watch type configurations, you can use the *show dialer* command:

```
Router1#show dialer
```

For dial configurations that use the backup interface configuration, you can use the *show backup* command:

```
Router1#show backup
```

And, for backup configurations that use ISDN, you can get some additional information from the *show isdn status*, *show isdn active*, and *show isdn history* commands:

```
Router1#show isdn status
Router1#show isdn active
Router1#show isdn history
```

13.7.3 Discussion

The *show dialer* command provides a lot of useful information about existing dial sessions as well as some historical statistics:

```
Router1#show dialer
BRI0 - dialer type = ISDN

Dial String      Successes   Failures    Last DNIS   Last status
0 incoming call(s) have been screened.
0 incoming call(s) rejected for callback.

BRI0:1 - dialer type = ISDN
Idle timer (300 secs), Fast idle timer (20 secs)
Wait for carrier (30 secs), Re-enable (15 secs)
Dialer state is data link layer up
Dial reason: ip (s=10.1.99.55, d=224.0.0.10)
```

```
Interface bound to profile Dialer1
Current call connected 00:03:18
Connected to 95551212 (dialhost)

BRI0:2 - dialer type = ISDN
Idle timer (120 secs), Fast idle timer (20 secs)
Wait for carrier (30 secs), Re-enable (15 secs)
Dialer state is idle

Dialer1 - dialer type = DIALER PROFILE
Load threshold for dialing additional calls is 100
Idle timer (300 secs), Fast idle timer (20 secs)
Wait for carrier (30 secs), Re-enable (15 secs)
Dialer state is data link layer up
Number of active calls = 1
Number of active circuit switched calls = 0

Dial String      Successes   Failures    Last DNIS   Last status
95551212               2          0    00:03:19       successful   Default
Router1#
```

There is a lot of useful information in this output. First, notice that there is an active dial session on the first B-channel of this ISDN BRI interface, BRI0:1. It has been connected for a little more than three minutes, and you can see the dial string that represents the remote telephone number. The second ISDN B-channel, BRI0:2, is not connected, presumably because the router has yet not seen the minimum traffic threshold that we specified for bringing up the second channel, or perhaps because it isn't configured for PPP multilink.

But there is another extremely important piece of information here. Note the line marked "Dial reason," which shows the source and destination IP addresses of the packet that originally caused the router to start the dial session. In this case, the source IP address is `10.1.99.55`, which is the IP address of the dial interface itself. The destination IP address is `224.0.0.10`, which is extremely interesting because this is the multicast IP address that EIGRP uses to talk between routers. This is fine if we intended for this dial connection to remain up all the time. However, if this router was supposed to only dial when a primary link failed, looking at this output should tell you that the dialer list configuration is wrong.

The bottom of the display includes some historical information about each of the configured dial strings, and lists how often the router has been able to connect successfully using each string. In this case there is only one dial string, but if there were several, they would all appear with their respective totals.

The *show backup* command is only useful when you use the backup interface configuration, which is discussed in Recipe 13.4:

```
Router1#show backup

Primary Interface   Secondary Interface   Status
```

```
----------------   ------------------   ------
Serial0/0          BRI0/0               active backup
```

In this case the interface `BRI0/0` is operating as an active backup for the primary interface, `Serial0/0`. If the primary interface is working properly, the Status column will say "normal operation." In this case, the backup interface will go into a standby mode:

```
Router1#show int bri0/0
BRI0 is standby mode, line protocol is down
  Hardware is BRI
   Internet address is 10.1.99.55/24
  MTU 1500 bytes, BW 64 Kbit, DLY 20000 usec,
      reliability 255/255, txload 1/255, rxload 1/255
  Encapsulation PPP, loopback not set
  Last input never, output never, output hang never
  Last clearing of "show interface" counters never
   Input queue: 0/75/0/0 (size/max/drops/flushes); Total output drops: 0
  Queueing strategy: weighted fair
   Output queue: 0/1000/64/0 (size/max total/threshold/drops)
      Conversations  0/0/16 (active/max active/max total)
      Reserved Conversations 0/0 (allocated/max allocated)
      Available Bandwidth 48 kilobits/sec
  5 minute input rate 0 bits/sec, 0 packets/sec
  5 minute output rate 0 bits/sec, 0 packets/sec
     0 packets input, 0 bytes, 0 no buffer
      Received 0 broadcasts, 0 runts, 0 giants, 0 throttles
     0 input errors, 0 CRC, 0 frame, 0 overrun, 0 ignored, 0 abort
     0 packets output, 0 bytes, 0 underruns
      0 output errors, 0 collisions, 7 interface resets
      0 output buffer failures, 0 output buffers swapped out
     0 carrier transitions
Router1#
```

If you are using ISDN for dial backup in any of the configurations discussed, you can get other useful information through the various *show isdn* commands:

```
Router1#show isdn status
Global ISDN Switchtype = basic-ni
ISDN BRI0/0 interface
     dsl 0, interface ISDN Switchtype = basic-ni
    Layer 1 Status:
     ACTIVE
    Layer 2 Status:
     TEI = 89, Ces = 1, SAPI = 0, State = MULTIPLE_FRAME_ESTABLISHED
     TEI = 90, Ces = 2, SAPI = 0, State = MULTIPLE_FRAME_ESTABLISHED
     TEI 89, ces = 1, state = 8(established)
         spid1 configured, spid1 sent, spid1 valid
         Endpoint ID Info: epsf = 0, usid = 70, tid = 1
     TEI 90, ces = 2, state = 8(established)
         spid2 configured, spid2 sent, spid2 valid
         Endpoint ID Info: epsf = 0, usid = 71, tid = 2
```

```
    Layer 3 Status:
     1 Active Layer 3 Call(s)
    Activated dsl 0 CCBs = 1
      CCB:callid=801A, sapi=0, ces=1, B-chan=1, calltype=DATA
    The Free Channel Mask:  0x80000002
Total Allocated ISDN CCBs = 2
Router1#
```

This example shows a single active call on one of the ISDN B-channels. Another useful piece of information is provided by the lines that say that each SPID was configured, sent, and considered "valid" by the switch. In this case, we were connected to a *basic-ni* type switch, which requires us to manually configure SPIDs. So it is important to check that the switch accepted these values.

If there are active calls, the output of *show isdn active* can also be useful:

```
Router1#show isdn active
-------------------------------------------------------------------------------
                                ISDN ACTIVE CALLS
-------------------------------------------------------------------------------
History table has a maximum of 100 entries.
History table data is retained for a maximum of 15 Minutes.
-------------------------------------------------------------------------------
Call    Calling      Called       Remote  Seconds Seconds Seconds Charges
Type    Number       Number       Name    Used    Left    Idle    Units/Currency
-------------------------------------------------------------------------------
Out             +95551212       dialhost      207                0      0
-------------------------------------------------------------------------------

Router1#
```

Here you can see that there is a single active call. This output also tells you exactly how long the call has been connected, and whether it was an in or outbound connection. You can also get some potentially useful historical information about previous calls from the *show isdn history* command:

```
Router1#show isdn history
-------------------------------------------------------------------------------
                                ISDN CALL HISTORY
-------------------------------------------------------------------------------
History table has a maximum of 100 entries.
History table data is retained for a maximum of 15 Minutes.
-------------------------------------------------------------------------------
Call    Calling      Called       Remote  Seconds Seconds Seconds Charges
Type    Number       Number       Name    Used    Left    Idle    Units/Currency
-------------------------------------------------------------------------------
Out                                      Failed
Out             +95551212       dialhost       20                      0
Out             +95551212       dialhost      219                0      0
-------------------------------------------------------------------------------

Router1#
```

This router has attempted to make three outbound calls. The first call failed, the second lasted 20 seconds before disconnecting, and the third lasted 219 seconds. This output can be particularly useful if you think that your router might be dialing too often, or that it might be frequently dialing and dropping calls.
You can also look in the router's logging buffer for log messages. By default, every time the router dials, it will generate at least one log message. Please refer to Chapter 18 for more information on logging.

Top

Recipe 13.8 Debugging Dial Backup

13.8.1 Problem

Your dial backup is not behaving properly and you want to debug it to isolate and resolve the problem.

13.8.2 Solution

The most common reasons for failed dial backup calls are incorrect dial strings and PPP authentication problems. You can easily diagnose both of these problems with this command:

```
Router1#debug ppp authentication
```

Here is another useful command for diagnosing problems with dialer configurations:

```
Router1#debug dialer
```

13.8.3 Discussion

When you use CHAP authentication with PPP, as we have done throughout this chapter, it is relatively easy to debug most common problems. We like to use the *debug ppp authentication* command because it pinpoints the most frequent problems:

```
Jun 28 14:04:05.211: BR0/0:1 PPP: Phase is AUTHENTICATING, by both
Jun 28 14:04:05.211: BR0/0:1 CHAP: O CHALLENGE id 1 len 33 from "Router1"
Jun 28 14:04:05.211: BR0/0:1 AUTH: Started process 0 pid 60
Jun 28 14:04:05.235: BR0/0:1 CHAP: I CHALLENGE id 35 len 33 from "dialhost"
Jun 28 14:04:05.235: BR0/0:1 CHAP: O RESPONSE id 35 len 33 from "Router1"
Jun 28 14:04:05.267: BR0/0:1 CHAP: I SUCCESS id 35 len 4
Jun 28 14:04:05.271: BR0/0:1 CHAP: I RESPONSE id 1 len 33 from "dialhost"
Jun 28 14:04:05.271: BR0/0:1 CHAP: O SUCCESS id 1 len 4
```

This shows the PPP authentication challenge and response handshake between two routers. This is one of the first places to look when you have dialup sessions that refuse to connect. In this particular trace, the router Router1 has dialed to dialhost. Upon connection, both routers present a password challenge. In this case, you can see that both routers responded correctly, and authentication was successful. You should see a similar trace on both routers if everything is working properly.

If you enable this debug command and see nothing, then you know that the routers are not reaching this point, indicating that there is a problem at a lower layer. It would then be a good idea to start by making

sure that you have the correct dial string, which you can verify by simply looking at the state of the dial interface on the receiving end to see if it picks up the phone when you call from the other end.

Another useful command to see if your router is dialing appropriately is *debug dialer*:

```
Jun 28 14:04:02.691: BR0/0 DDR: rotor dialout [priority]
Jun 28 14:04:02.691: BR0/0 DDR: Dialing cause ip (s=10.1.99.55, d=10.1.99.1)
Jun 28 14:04:02.691: BR0/0 DDR: Attempting to dial 95551212
Jun 28 14:04:05.311: Di1 DDR: dialer protocol up
```

Here you can see that the router is dialing because of an IP packet, and the output shows the source and destination IP addresses of this packet. It also shows the dial string, the interface, and the fact that it connected successfully.

Top

Chapter 14. NTP and Time

Top

Introduction

Many engineers overlook the importance of accurate timekeeping on a router. It is often extremely useful to be able to accurately pinpoint when a particular event occurred. You may want to compare network event messages from various routers on your network for fault isolation, troubleshooting, and security purposes. This is impossible if their clocks are not set to a common source. In fact, merely setting the clocks to a single common standard is not enough, because some clocks run a little bit fast and others run a little bit slow. So router clocks need to be continuously adjusted and synchronized.

Network Time Protocol (NTP) is the de facto standard for Internet time synchronization. The current standard for NTP is Version 3, which is defined in RFC 1305. The IETF is currently developing a new version.

The protocol allows devices to communicate over UDP port 123 to obtain time from an authoritative time source such as a radio clock, atomic clock, or GPS-based time source. An NTP server connected directly to one of these known reliable time sources is called a *Stratum 1* timeserver. Stratum 2 timeservers receive their time via NTP from a Stratum 1 server, and so forth, up to a maximum of Stratum 16. Stratum numbers are analogous to hop counts from the authoritative time source. NTP generally prefers lower stratum servers to higher stratum servers unless the lower stratum server's time is significantly different.

The algorithm is able to detect when a time source is likely to be extremely inaccurate, or *insane*, and to prevent synchronization in these cases, even if the inaccurate clock is at a lower stratum level. And it will never synchronize a device to another server that is not synchronized itself.

The NTP protocol is extremely efficient and lightweight. It can synchronize a client device's clock with the server device's clock to within milliseconds, while exchanging packets as rarely as once every 1024 seconds (roughly 17 minutes). Even over WAN links, NTP is able to synchronize clocks to within tens of milliseconds. To achieve this, it has algorithms that estimate and reduce the affects of network jitter and latency. It is also able to use multiple time sources simultaneously for improved reliability and fault tolerance.

As the multiple stratum levels suggest, NTP uses a hierarchical topology. However, this is relevant only to the relationships between clients and servers, which do not need to be physically adjacent on the network. The protocol does not require any particular underlying network topology. NTP Version 3 has three different operational modes: master/slave (server/client), symmetric (peers), and a broadcast mode in which the clients passively listen for updates from a server. Some implementations also have a multicast mode that most likely foreshadows some of what will be in Version 4. Cisco has recently added multicast support, which we discuss in Recipe 14.10.

In the master/slave mode, the client device periodically sends a message to one or more servers to request synchronization. Because the server is closer to the original time source, its clock is assumed to be more reliable. So the server will synchronize the client's time, but will not allow the client to change its own clock. The server passively listens for these synchronization requests from clients.

In the symmetric peer-to-peer mode, both NTP devices synchronize one another. Peers can operate in active or passive mode. However, at least one of a pair of peers must be active or nobody will ever start the conversation.

The broadcast and multicast modes of operation are used to synchronize a large number of passive client devices in a network. This has the advantage of saving bandwidth caused by multiple requests for synchronization. In most cases, the overhead caused by every device making separate requests is minimal, however. The broadcast and muliticast modes have the disadvantage of being less precise than a poll-response model because there is no way for the client device to estimate network latency. The multicast mode is somewhat more useful than the broadcast mode because it allows you to synchronize devices on many network segments from a single source. However, multicast routing must be enabled on the network. We discuss multicast routing in Chapter 23.

The NTP client and server software runs on most modern operating systems including Unix, Windows, and Mac OS. You can find source code and binary executable NTP software for various operating systems at http://www.eecis.udel.edu/~ntp/software/index.html. The general information web page for all things related to NTP and the ongoing protocol and software development is http://www.ntp.org.

Organizations can purchase their own authoritative time sources or obtain time services via the Internet. There are small, cost-effective GPS Stratum 1 servers on the market today, which you can use as an extremely accurate reference clock. These devices typically cost a few thousand dollars and can be easily rack-mounted in a computer room in the core of your network. Alternatively, there are hundreds of public Stratum 1 and 2 timeservers available on the Internet that allow devices to connect and synchronize with them free of charge. Sending synchronization signals through the public Internet introduces some additional jitter that is somewhat more difficult to estimate. So this method is slightly less accurate than using your own timeserver, but the difference is rarely more than a few milliseconds, and you can reduce the impact of this problem by synchronizing with multiple servers. For most applications, the publicly available servers are more than adequate.

The web site http://www.eecis.udel.edu/~mills/ntp/servers.htm has useful information about public NTP servers available through the Internet.

Top

Recipe 14.1 Timestamping Router Logs

14.1.1 Problem

You want the router to record the time along with log and debug messages.

14.1.2 Solution

The *service timestamp* global configuration command enables timestamps on debug and logging messages. Use the *log* keyword to turn on timestamping of log messages:

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#service timestamps log datetime localtime
Router(config)#end
Router#
```

The command to turn on timestamps for debug messages is similar, but uses the *debug* keyword:

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#service timestamps debug datetime localtime
Router(config)#end
Router#
```

14.1.3 Discussion

By default, Cisco routers create log and debug messages without any form of timestamp. These messages are useful if the administrator is watching them in real- time. However, it impossible to look at the logs later and understand what exactly happened when. The following example illustrates the problem:

```
%SYS-5-CONFIG_I: Configured from console on vty1 (172.25.1.1)
%CLEAR-5-COUNTERS: Clear counter on all interfaces on vty1 (172.25.1.1)
%OSPF-5-ADJCHG: Process 55, Nbr 172.25.25.6 on FastEthernet0/0.1 from FULL to DOWN
```

You can't tell when the router configuration was changed or when the OSPF neighbor state changed. And, in particular, you can't tell if the configuration change caused the OSPF problem. Having accurately timestamped log messages makes troubleshooting and problem determination much easier.

The example below shows the same incident with timestamps enabled:

```
Mar  9 04:43:31: %SYS-5-CONFIG_I: Configured from console on vty1 (172.25.1.1)
Mar  9 04:44:17: %CLEAR-5-COUNTERS: Clear counter on all interfaces on vty1 (172.25.
```

```
1.1)
Mar 11 06:19:27: %OSPF-5-ADJCHG: Process 55, Nbr 172.25.25.6 on FastEthernet0/0.1
from FULL to DOWN
```

In the previous example, it is clear that the router configuration changed on March 9 at 04:44:17, but that the OSPF problem happened two days later. Hopefully this example makes it clear why we strongly advocate enabling this feature on all routers.

The same issues are generally true with debug messages, except for two things. First, most network administrators tend to turn on debug only when it is required for troubleshooting a particular problem, so the problem isn't what hour or day something happened, but rather what millisecond. The second problem is that, when troubleshooting a complex connectivity problem, it is often necessary to have the clocks synchronized extremely closely. We will deal with this second problem in Recipe 14.5 when we talk about NTP. However, the first problem means that the router has to display debugging messages with more precise timestamps than log messages.

You can enable millisecond timestamps for debug messages using the *msec* keyword as follows:

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#service timestamps debug datetime localtime show-timezone msec
Router(config)#end
Router#
```

The resulting timestamps on debugging messages look like this:

```
Mar  9 04:44:39.009: IP: s=172.25.1.5 (local), d=172.25.1.1 FastEthernet0/0.1), len
65, sending
Mar  9 04:44:39.177: IP: s=172.25.1.5 (local), d=172.25.1.3 (FastEthernet0/0.1), len
65, sending
Mar  9 04:44:39.341: IP: s=172.25.1.3 (FastEthernet0/0.1), d=172.25.1.5
(FastEthernet0/0.1), len 111, rcvd 3
```

The *msec* keyword is also available for log timestamps, although it tends to be less useful.

For large WANs that span several time zones, it is useful to enable local time zone information as part of the timestamp as well. This is done using the *show-timezone* keyword:

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#service timestamps log datetime localtime show-timezone
Router(config)#end
Router#

Mar  9 13:47:03 EST: %SYS-5-CONFIG_I: Configured from console on vty1 (172.25.1.1)
```

Instead of using absolute time, with the *date-time* keyword, you can use the *uptime* keyword to force the router to display log timestamps based on how long the router had been up when the event occurred:

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#service timestamps log uptime
Router(config)#end
Router#
```

This gives a log messages like this:

```
2d04h: %CLEAR-5-COUNTERS: Clear counter on all interfaces on vty2 (172.25.1.1)
```

This message indicates that somebody cleared the router's counters when the router had been active for two days and four hours. If you want to correlate this to a real clock time, you'll have to look at the router's current uptime and subtract to find out when the event occurred. This is rarely as useful as the absolute date timestamps because it requires too much effort to correlate log messages from different routers. It is also considerably less accurate because of the uncertainty in figuring out the precise time that the router restarted. This sort of timestamping is useful only if you don't intend to set the clock to an accurate absolute time.

Top

Recipe 14.2 Setting the Time

14.2.1 Problem

You want to set the clock on the router.

14.2.2 Solution

You can set the internal system clock using the *clock set* in enable mode:

```
Router#clock set 14:27:22 March 9 2003
```

Some high-end routers, such as the 4500 series, 7000 series, 7200 series, and 7500 series, have a battery-protected *calendar* function that continues to keep time even if the router is temporarily powered off. You can set this calendar function using the *calendar set* command in enable mode:

```
Router#calendar set 14:34:39 March 9 2003
```

In both cases, the router will accept either "hh:mm:ss day month year" or "hh:mm:ss month day year" notation.

14.2.3 Discussion

Every Cisco router has an internal system clock. When the router boots, the internal system clock starts to maintain the current date and time. If there is no battery-protected calendar in the router, the clock will start with a default initial value of Monday March 1, 1993 at midnight. If you want accurate time, you need to set it manually as described earlier, or use the automated method given in Recipe 14.5.

As we said, most high-end routers have an internal battery-powered clock called a calendar. Router calendars are able to maintain accurate time and date information, even during power interruptions. When the router initializes, it automatically synchronizes the internal system clock with the date stored with the calendar.

You can view the current calendar time using the following command:

```
Router>show calendar
14:34:39 UTC Sat Mar 9 2003
Router>
```

If your router returns an error message when you issue this command, then it does not contain a calendar:

```
Router#show calendar
                  ^
% Invalid input detected at '^' marker.

Router#
```

Note, however, that the "clock" time and the "calendar" time are kept on different clocks that may differ at any given moment. After router initialization, these two clocks may drift apart or be set independently of one another. Fortunately, Cisco's IOS does provide methods of synchronizing the two time sources after initial power up.

To set the calendar to the internal clock time, use the following command in enable mode:

```
Router#clock update-calendar
Router#
```

You can also set the internal clock to the calendar time using the command:

```
Router#clock read-calendar
Router#
```

Both the internal system clock and calendar use 24-hour time notation rather than 12-hour A.M./P.M. notation.

Top

Recipe 14.3 Setting the Time Zone

14.3.1 Problem

You want to change the time zone on the router.

14.3.2 Solution

To configure the router's local time zone, use the following configuration command:

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#clock timezone EST -5
Router(config)#end
Router#
```

The *clock timezone* configuration command accepts any freeform zone name (EST, PST, Eastern, etc.) followed by an offset from the UTC (-24 to 24 hours), and an optional offset from UTC in minutes for areas that require it.

14.3.3 Discussion

By default, the router uses UTC, also called Coordinated Universal Time. UTC, formerly known as Greenwich Mean Time (GMT), has become the worldwide standard for time and date. The principles for calculating local time zones from UTC are the same as for GMT. The only difference is that UTC is based on precise atomic clocks, shortwave radio signals, and satellites to ensure accuracy. These devices do not actually need to be located in Greenwich, England.

It is useful to set the router's internal system clock to display the time in the local time zone. North American clocks set to UTC, for example, display the time between five and eight hours ahead of the local time. This means that somebody reading the clock has to do some mental arithmetic to translate to local time, which is sometimes awkward and makes correlating the times of network problems more difficult than it needs to be.

You can view the current time zone information with the *show clock detail* command:

```
Router>show clock detail
14:27:31.415 EST Wed Mar 9 2003
Time source is NTP
```

```
Router>
```

Many organizations choose to configure all of their routers to the same time zone to make problem correlation easier, regardless of router location. The network administrators will configure all of their routers to the same time zone, even if they are physically located in a different part of the world. We recommend doing this because it simplifies troubleshooting by eliminating the need to do a lot of mental arithmetic that can unnecessarily slow down an already difficult and stressful situation.

Table 14-1 shows the configuration for several of the most commonly used time zones in North America.

Table 14-1. North American time zones

Time Zone	Abbr.	Offset from UTC	Configuration Command
Hawaiian Standard Time	HST	UTC -10	clock timezone HST -10
Alaska Standard Time	AKST	UTC -9	clock timezone AKST -9
Pacific Standard Time	PST	UTC -8	clock timezone PST -8
Mountain Standard Time	MST	UTC -7	clock timezone MST -7
Central Standard Time	CST	UTC -6	clock timezone CST -6
Eastern Standard Time	EST	UTC -5	clock timezone EST -5
Atlantic Standard Time	AST	UTC -4	clock timezone AST -4
Newfoundland Standard Time	NST	UTC -3.5	clock timezone NST -3 30

14.3.4 See Also

Recipe 14.4; Recipe 14.5

Top

Recipe 14.4 Adjusting for Daylight Saving Time

14.4.1 Problem

You want the router to automatically adjust to Daylight Saving Time.

14.4.2 Solution

Some areas, such as most of North America and Europe, have consistent and common rules for when to switch between winter or Standard time and summer or Daylight Saving Time. The North American rule, for those areas that observe Daylight Saving Time, is to move an hour ahead at 2:00 A.M. on the first Sunday in April, and back an hour at 2:00 A.M. on the last Sunday in October. This is the default for Cisco routers that have been configured for summer time:

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#clock summer-time EDT recurring
Router(config)#end
Router#
```

You can also specify exact recurrence rules. You could use the following command to explicitly define the North American rules:

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#clock summer-time EDT recurring first sun apr 02:00 last sun oct 02:00
Router(config)#end
Router#
```

You can modify this line to represent other standard recurrence rules. For example, in the southern hemisphere, Daylight Saving Time commences towards the end of the year to match their summer season. To define the rules for Australian Eastern Daylight Time (AEDT), you could use the following command:

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#clock summer-time AEDT recurring last sun oct 02:00 last sun mar 02:00
Router(config)#end
Router#
```

Some areas do not have consistent daylight saving rules. In these areas, you must configure the start and end dates explicitly:

```
Router#configure terminal
```

```
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#clock summer-time EDT date 26 oct 2003 02:00 6 apr 2003 02:00
Router(config)#end
Router#
```

14.4.3 Discussion

The router will not observe Daylight Saving Time by default. The *clock summer-time* command allows you to automatically reset the router's clock each fall and winter so that you don't have to do so manually, which can be a daunting task in a large network.

You can see how the router plans to implement the change to summertime with the *show clock detail* command:

```
Router>show clock detail
14:05:04.299 EST Sun Mar 10 2003
Time source is user configuration
Summer time starts 02:00:00 EST Sun Apr 6 2003
Summer time ends 02:00:00 EDT Sun Oct 26 2003
Router>
```

In this case, the router is set to the default North American Daylight Saving Time start and end dates.

14.4.4 See Also

Recipe 14.3

Top

Recipe 14.5 Synchronizing the Time on All Routers (NTP)

14.5.1 Problem

You want your routers to automatically learn the time and synchronize their clocks through the network.

14.5.2 Solution

Network Time Protocol (NTP) is an open standard protocol for time synchronization. You can implement NTP on a router to provide automatic and efficient time synchronization. To enable a basic NTP configuration, enter the following commands:

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#clock timezone EST -5
Router(config)#clock summer-time EDT recurring
Router(config)#ntp server 172.25.1.1
Router(config)#end
Router#
```

The *ntp server* command accepts either IP addresses or hostnames. To use a hostname, however, you will need to configure the router to either use a static host table or DNS for name resolution, as discussed in Chapter 2.

Some low-end routers, such as the Cisco 1000 series, Cisco 1600 series, Cisco 1720 series, and Cisco 1750 series do not support NTP. For these, Cisco provides support for the Simple Network Time Protocol (SNTP), which is a compatible subset of the NTP standard. The SNTP configuration is similar to NTP:

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#clock timezone EST -5
Router(config)#clock summer-time EDT recurring
Router(config)#sntp server 172.25.1.1
Router(config)#end
Router#
```

14.5.3 Discussion

When NTP is enabled on a router, it will start trying to synchronize with the configured peers or servers as soon as it boots. By default, the router's clock always displays the time in the UTC time zone. So we recommend configuring an appropriate local time zone as in this example, and shown in more detail in Recipe 14.3 and Recipe 14.4.

Most Cisco routers fully support NTP Versions 1, 2, and 3, and also include some features such as multicast support that are not yet fully standard. There are actually no important protocol differences between the three versions, and they operate together well. The main differences between them are apparent in things such as the algorithms used for estimating latency. Later versions also offer some additional modes of operation.

Version 3 of the NTP protocol has several different modes of operation. A device can be a client, server, peer, multicast client or server, or a broadcast client or server. Once a router has built an NTP association and synchronized its clock, it automatically becomes a fully functional NTP server itself, capable of providing NTP services to other NTP clients.

By default, the source IP address that a router uses for its NTP packets will be the address of the interface that sends them. This is usually not a problem. However, in networks with many redundant paths, it is possible to have a router suddenly change the interface that it uses to communicate with another NTP device simply because the routing tables changed. If the other device is configured to accept only a limited number of connections or if it has rules allowing connections only from certain specified devices, then NTP might break.

To get around these sorts of problems, Cisco provides two methods for manually assigning the source address of NTP packets. The first is a global command that affects all NTP packets, and the second sets different source addresses for different NTP associations.

The global command assigns a source IP address for all associations, even the ones that the router passively accepts:

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#ntp source loopback0
Router(config)#end
Router#
```

This example tells NTP to use the IP address of the `loopback0` interface as the source address for all NTP associations.

Sometimes you want the router to use different source addresses for different servers:

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#ntp server 172.25.1.1 source FastEthernet 0/0.1
Router(config)#ntp server 10.1.1.1 source Serial 0/0
Router(config)#end
```

```
Router#
```

Assigning a source address for one NTP association like this does not affect other NTP associations on the router. You can assign the global command and the per-association command at the same time, and the router will use the global address for everything except the specifically defined associations.

In Recipe 14.2, we mentioned that many high-end routers contain battery-protected calendars that operate independently from the main system clock. By default, NTP will set only the system clock. But you can also synchronize the calendar with NTP using the *ntp update-calendar* command:

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#ntp update-calendar
Router(config)#end
Router#
```

Two other useful timestamps are automatically enabled on routers that have their clocks synchronized with NTP. First, the *show version* command gives the exact time when the router last initialized:

```
Router#show version
Cisco Internetwork Operating System Software
IOS (tm) C2600 Software (C2600-JK9O3S-M), Version 12.2(7a), RELEASE SOFTWARE (fc2)
Copyright (c) 1986-2002 by cisco Systems, Inc.
Compiled Thu 21-Feb-02 03:48 by pwade
Image text-base: 0x80008088, data-base: 0x8153F5D0

ROM: System Bootstrap, Version 11.3(2)XA4, RELEASE SOFTWARE (fc1)

router uptime is 3 days, 2 hours, 7 minutes
System returned to ROM by power-on
System restarted at 20:56:01 EST Sat Mar 8 2003
System image file is "flash:c2600-jk9o3s-mz.122-7a.bin"
<removed>
```

Second, the *show running-config* command gives a timestamp of when the configuration last changed and when the running configuration was last saved to NVRAM:

```
Router#show running-config
Building configuration...

Current configuration : 3353 bytes
!
! Last configuration change at 21:25:52 EST Tue Mar 11 2003 by ijbrown
! NVRAM config last updated at 22:13:48 EST Sat Mar 8 2003 by kdooley
!
version 12.2
service timestamps debug datetime msec
service timestamps log datetime localtime
service password-encryption
service compress-config
```

```
<removed>
```

SNTP is another UDP-based time synchronization protocol that is essentially a simplified version of NTP that only supports client time synchronization. Several of Cisco's low-end routers support only SNTP and cannot synchronize the clocks of other devices.

Since SNTP is a subset of NTP, it allows the router to synchronize to central NTP servers, and it can use NTP broadcast messages as well. SNTP is much less accurate than NTP, generally synchronizing clocks only to within 100 milliseconds (a tenth of a second) of one another. SNTP-based routers can obtain time services from multiple NTP sources, but SNTP lacks the ability to make intelligent server decisions (unlike NTP). If the router is configured with several servers, SNTP will simply choose the one with the lowest NTP stratum number. If it knows about two servers that are both at the same stratum level, the router chooses the one that sends the first packet. SNTP will select an NTP server with a higher stratum only if a lower stratum server becomes unreachable.

There are only two SNTP configuration options. The router can communicate directly with a server, or you can configure it to listen for NTP broadcasts:

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#sntp ?
  broadcast  Configure SNTP broadcast services
  server     Configure SNTP server
Router(config)#end
Router#
```

You can view the SNTP status on the router with the *show sntp* command:

```
Router>show sntp
SNTP server     Stratum   Version    Last Receive
172.25.1.1         2         3        00:00:24    Synced
172.25.1.3         2         3        00:00:51
Router>
```

Top

Recipe 14.6 Configuring NTP Redundancy

14.6.1 Problem

You want to configure more than one NTP server for redundancy.

14.6.2 Solution

You can improve NTP reliability by configuring several redundant servers. The reliability is better still if the router uses different paths to reach these servers:

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#clock timezone EST -5
Router(config)#clock summer-time EDT recurring
Router(config)#ntp server 172.25.1.1
Router(config)#ntp server 10.121.33.231
Router(config)#ntp peer 192.168.12.12
Router(config)#end
Router#
```

14.6.3 Discussion

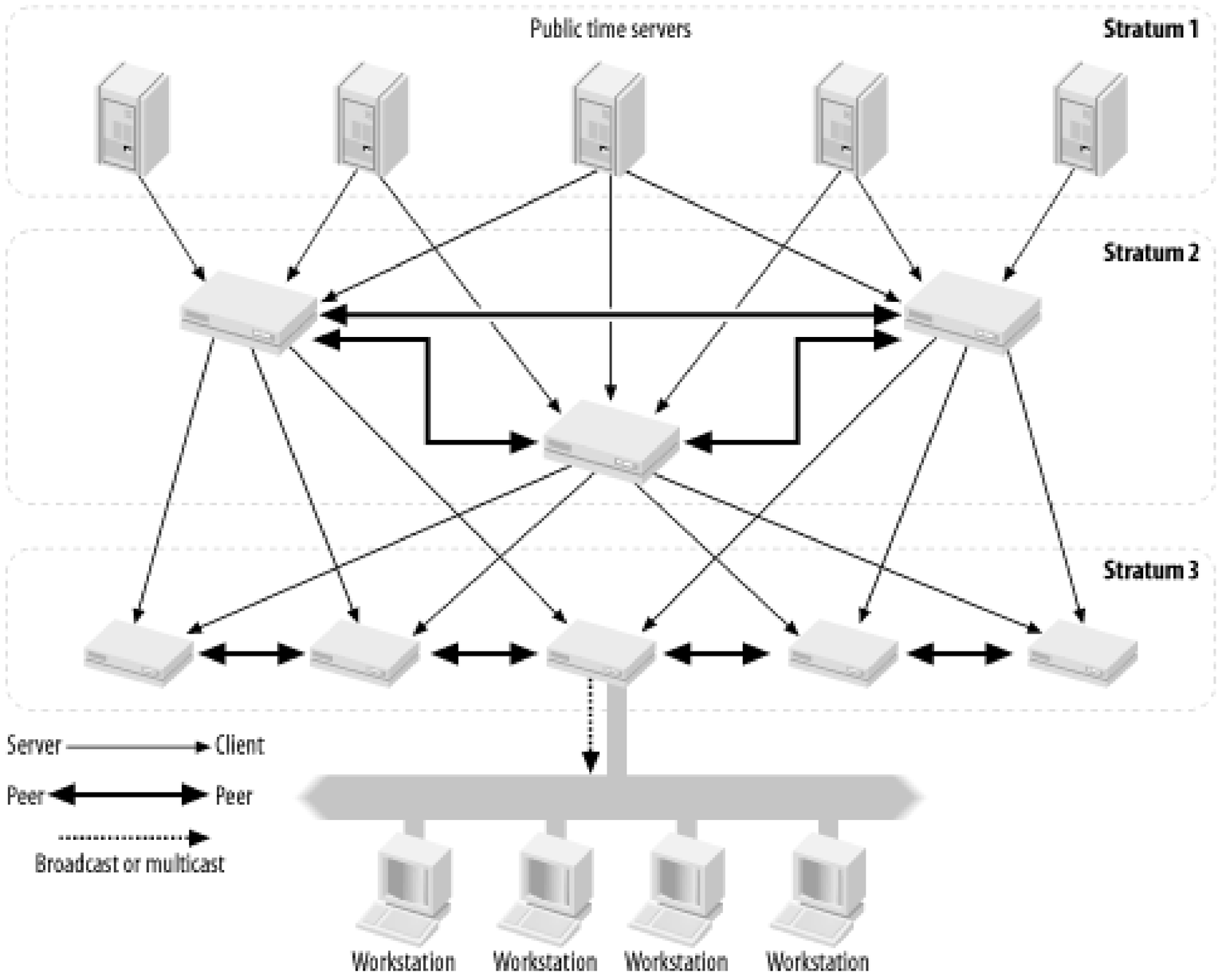
The NTP algorithms have built-in sanity checks to help choose the best time source. The NTP client chooses the most accurate time source and synchronizes its internal clock to that server. The algorithm continuously performs sanity checks to ensure that it synchronizes to the best possible server. It is also common for a router to change its preferred NTP server many times during a day.

Configuring multiple time sources improves reliability as well as the accuracy of a router's clock. Although NTP is a remarkably stable protocol, device and link failures can disrupt timing services to your router. Providing the router with a choice of NTP servers ensures accurate time synchronization and provides resilience in case of failure. Be sure to choose redundant NTP servers that provide alternate network paths and hardware.

An NTP network is a hierarchy of servers and clients configured in a redundant topology. At the top level, Stratum 1 NTP servers establish peer relationships with other Stratum 1 servers and server relationships to Stratum 2 servers. In turn, Stratum 2 servers peer symmetrically with other Stratum 2 servers, receive time feeds from one or more Stratum 1 servers, and act as servers for Stratum 3 devices. This pattern repeats to create an overall tree topology of stratum levels.

Figure 14-1 shows a typical NTP hierarchal topology with fully redundant paths and devices. The goal is to design an NTP hierarchy that can withstand a failure of any single networking entity, path, or device. Designing a resilient NTP topology requires little time and effort once you plot your time sources on an existing network diagram. Large networks tend to require more thought and effort when it comes to designing an overall NTP hierarchy; small networks can often make do with two NTP servers.

Figure 14-1. The NTP hierarchy



The important thing is to design the NTP topology to use its redundancy features. Once the NTP topology is defined, configuring the routers to participate in the hierarchy is straightforward. The previous example demonstrates a typical NTP configuration that incorporates redundancy features. The router configuration includes two NTP servers and an NTP peer:

```
Router>show ntp associations

      address         ref clock     st  when  poll reach  delay  offset    disp
*~172.25.1.1       192.5.41.40      2    57    64  377    30.0   60.32     1.2
+~10.121.33.231    192.5.41.209     2    11    64  377    30.0  -54.85     1.1
 ~192.168.12.12    172.25.1.1       3  4588  1024    0     4.9   58.07  16000.
*master (synced), # master (unsynced), + selected, - candidate, ~ configured
Router>
```

In this case, the router configuration includes multiple NTP time sources. The output indicates that one

of the time sources, `192.168.12.12`, is currently unreachable. Even though one of the configured time sources is unavailable, the router is unaffected because the other NTP associations remain up and synchronized. This means that accurate time services are uninterrupted by losing a single NTP time feed.

If your organization receives its NTP feed from the Internet, then it is highly recommended that you use at least two such NTP servers. To provide maximum stability, configure each of your servers with multiple NTP Stratum 1 servers and build a peer relationship between them. This ensures that your organization's time source remains as stable as possible.

Routers that lose connectivity to their Stratum 1 servers will rely on their own internal clocks until service is restored. Similarly, routers that are isolated from their upstream stratum servers also rely on their own internal clocks. However, in these situations, router internal clocks will not be synchronized to each other, meaning the network clocks will drift further apart until connectivity returns. NTP generally tolerates interruptions of less than an hour quite well.

Top

Recipe 14.7 Setting the Router as the NTP Master for the Network

14.7.1 Problem

You want to use the router as an NTP server to act as the primary time source for the network.

14.7.2 Solution

There is no need for a dedicated NTP server; you can pick one or two routers to act as authoritative NTP servers for the whole network. (The router should have a calendar function.)

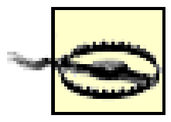
```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#clock timezone EST -5
Router(config)#clock summer-time EDT recurring
Router(config)#clock calendar-valid
Router(config)#ntp master 8
Router(config)#end
Router#
```

14.7.3 Discussion

When no authoritative time sources are available, and you still need to synchronize clocks throughout a network, you can configure a router to act as an NTP master server. Although this situation should be considered as a last resort, the router can become a "self-professed" NTP master, even if it is not actually synchronized from an accurate clock.

NTP provides two important services: accurate time setting and clock synchronization. Enabling a router to become an NTP master will not guarantee accurate time, but it will ensure that all network components' time remain synchronized. This distinction is important because routers acting as NTP masters can synchronize their entire network to an incorrect time. However, if the reliable time sources that your network uses become unreachable for any length of time, it is often useful to at least keep the clocks synchronized, even if they aren't completely accurate.

For this reason, we recommend that if your routers provide master NTP services, they should never advertise themselves as Stratum 1 servers. Timekeeping instability can occur on networks with both legitimate NTP time sources and router NTP masters. This is because NTP clients cannot distinguish between legitimate time sources and a router acting as a NTP master. Please use caution when configuring this feature.



Setting a router to act as a NTP master can interrupt legitimate time sources.

In the example, we have set the NTP master to the relatively safe Stratum level 8. In most cases, this will prevent the router from being preferred over valid time sources since NTP clients tend to synchronize to the lowest available Stratum server, assuming their clock is sane.

For obvious reasons, only routers with battery-protected timers or calendars are good candidates to become NTP masters. Otherwise, a power failure or reload of a single router could cause the entire network to become unsynchronized. For increased resilience, the design should include a minimum of two NTP master routers with peering configured between them.

14.7.4 See Also

Recipe 14.3; Recipe 14.4; Recipe 14.5

Top

Recipe 14.8 Changing NTP Synchronization Periods

14.8.1 Problem

You want to adjust how often routers send NTP packets to verify clock synchronization.

14.8.2 Solution

You cannot manually change NTP's polling rates. The protocol has an adaptive algorithm that automatically adjusts the polling interval.

14.8.3 Discussion

NTP is an extremely efficient protocol that actively monitors all aspects of network timing to adjust its configuration accordingly. Upon initialization of NTP, a router sets its cycle to poll about once every 64 seconds. As the local clock becomes synchronized and stable, the router will adaptively back off the poll cycle to a maximum of 1024 seconds (roughly 17 minutes):

```
Router>show ntp associations

      address         ref clock     st  when   poll reach  delay  offset    disp
*~172.25.1.1       130.207.244.240   2   440   1024  377     1.6   -3.23     5.6
+~172.25.1.3       204.152.184.72    2   829   1024  377     1.7    8.06     0.9
* master (synced), # master (unsynced), + selected, - candidate, ~ configured
Router>
```

Note that the poll cycle for the two configured servers has been throttled back to 1024 seconds. This indicates that the router is in a stable network and the time services are consistently accurate. This also illustrates the remarkable efficiency of NTP.

Top

Recipe 14.9 Using NTP to Send Periodic Broadcast Time Updates

14.9.1 Problem

You want to set up your router to use the NTP broadcast mode so that devices do not need to query periodically for the time.

14.9.2 Solution

Use the NTP broadcast interface configuration command to enable NTP server broadcast:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#clock timezone EST -5
Router1(config)#clock summer-time EDT recurring
Router1(config)#ntp server 172.25.1.1
Router1(config)#ntp server 172.25.1.2
Router1(config)#interface FastEthernet0/0
Router1(config-if)#ntp broadcast
Router1(config-if)#end
Router1#
```

To enable an NTP broadcast client on the router, enter the following:

```
Router2#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router2(config)#clock timezone EST -5
Router2(config)#clock summer-time EDT recurring
Router2(config)#ntp broadcastdelay 4
Router2(config)#interface Ethernet0
Router2(config-if)#ntp broadcast client
Router2(config-if)#end
Router2#
```

14.9.3 Discussion

NTP associations are usually configured in a master/slave relationship, but the server (router) can also send periodic time updates using broadcast messages. This is useful on LAN segments that contain a large number of devices requiring NTP synchronization. Instead of responding to a large number of *unicast* NTP packets through a single interface, the router can simply send a single broadcast packet at a regular interval.

Devices configured to accept NTP broadcast messages can synchronize their internal clocks without ever sending a single NTP request packet. However, this simplicity comes at the cost of reduced timing accuracy since the traffic only flows in one direction. The accuracy improves slightly by configuring an estimated broadcast delay on the client side, using the *ntp broadcastdelay* configuration command, as in the client configuration example.

Devices whose clocks are synchronized with NTP broadcasts are usually accurate to within a few hundreds of milliseconds. This is more than adequate for most client workstations. If some of the devices on the LAN require more accuracy, you can configure these devices with regular NTP associations as discussed in Recipe 14.5, Recipe 14.6, and Recipe 14.7. NTP broadcast mode does not prevent normal NTP client/server relationships from occurring as well, if required.

The next example shows a router configured as an NTP broadcast client that has synchronized its internal clock to a broadcast server:

```
Router2>show ntp associations detail
172.16.2.1 dynamic, our_master, sane, valid, stratum 3
ref ID 172.25.1.3, time C03A9BAB.5A1E7119 (22:46:51.352 EST Wed Mar 13 2003)
our mode bdcast client, peer mode bdcast, our poll intvl 64, peer poll intvl 64
root delay 116.56 msec, root disp 46.39, reach 376, sync dist 108.398
delay 5.19 msec, offset -0.3381 msec, dispersion 1.14
precision 2**16, version 3
org time C03A9C11.5A2376B6 (22:48:33.352 EST Wed Mar 13 2003)
rcv time C03A9C11.5B0B9E6E (22:48:33.355 EST Wed Mar 13 2003)
xmt time 00000000.00000000 (19:00:00.000 EST Thu Dec 31 1899)
filtdelay =     5.19    5.19    5.19    5.19    5.19    5.19    5.19    5.19
filtoffset =   -0.34   -0.53   -0.19   -0.18   -0.27   -0.32   -0.26   -0.34
filterror =     0.99    1.97    2.94    3.92    4.90    5.87    6.85    7.83
Router2>
```

Note that the *ntp broadcast* commands are interface configuration-level commands, configured on the interface that is sending or receiving the NTP broadcasts. However, the *ntp broadcastdelay* command is a global configuration command affecting all interfaces that use NTP broadcast features.

14.9.4 See also

Recipe 14.10; Recipe 14.12

Top

Recipe 14.10 Using NTP to Send Periodic Multicast Time Updates

14.10.1 Problem

You want to set up your router to use the NTP multicast mode so that devices do not need to query periodically for the time.

14.10.2 Solution

Use the *ntp multicast* interface command to allow the router to send NTP multicast packets:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#clock timezone EST -5
Router1(config)#clock summer-time EDT recurring
Router1(config)#ntp server 172.25.1.1
Router1(config)#ntp server 172.25.1.3
Router1(config)#interface FastEthernet 0/0
Router1(config-if)#ntp multicast 224.0.1.1 ttl 1
Router1(config-if)#end
Router1#
```

To enable NTP multicast client functionality on the router, use these commands:

```
Router2#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router2(config)#clock timezone EST -5
Router2(config)#clock summer-time EDT recurring
Router2(config)#interface Ethernet0
Router2(config-if)#ntp multicast client 224.0.1.1
Router2(config-if)#ntp multicast version 3
Router2(config-if)#end
Router2#
```

NTP multicast support is available starting in IOS Version 12.1.

14.10.3 Discussion

On the surface, the ability to forward NTP broadcast packets and NTP multicast packets on a LAN interface appear similar. However, there are some differences. First, NTP sends broadcast packets to the local broadcast address, `255.255.255.255`. This means that every device on the network must

examine the NTP packet. If there are devices on the network that are not NTP broadcast clients, then they will waste valuable system resources reading and discarding these NTP broadcast packets.

On the other hand, NTP multicast packets are sent to the well-known NTP multicast address (`224.0.1.1` by default), so only participating NTP multicast clients will examine these packets. The decision of whether to look at a multicast packet is made by the client device's Network Interface Card (NIC), which makes multicast traffic more efficient.

Second, broadcast packets never leave the local LAN segment or broadcast domain. However, multicast packets can be forwarded beyond the local segment via multicast routing, as discussed in Chapter 23. In the previous example, we have configured the server so that it sends these multicast packets with a Time-To-Live (TTL) value of 1. This effectively limits the range of the NTP multicast packets to the local segment, so you do not have to enable multicast routing. But we could choose to route the packet further by increasing the TTL value and enabling multicast routing.

Third, upon initial startup, multicast clients will forward several unicast NTP queries in quick succession to accurately estimate delay and jitter to the server. This allows multicast NTP clients to synchronize their clocks more accurately than broadcast clients. Once the initial packet exchanges occur, the client becomes completely passive and listens for the regularly scheduled NTP multicast server packets.

The following example shows the output of a network analyzer configured to capture all NTP packets on the wire:

```
07:36:15 172.25.1.5.ntp > 224.0.1.1.ntp:v3 mcast strat 3  [ttl 1]
07:37:19 172.25.1.5.ntp > 224.0.1.1.ntp:v3 mcast strat 3  [ttl 1]
07:38:23 172.25.1.5.ntp > 224.0.1.1.ntp:v3 mcast strat 3  [ttl 1]
07:39:27 172.25.1.5.ntp > 224.0.1.1.ntp:v3 mcast strat 3  [ttl 1]
07:40:31 172.25.1.5.ntp > 224.0.1.1.ntp:v3 mcast strat 3  [ttl 1]
07:41:35 172.25.1.5.ntp > 224.0.1.1.ntp:v3 mcast strat 3  [ttl 1]
07:42:39 172.25.1.5.ntp > 224.0.1.1.ntp:v3 mcast strat 3  [ttl 1]
07:43:43 172.25.1.5.ntp > 224.0.1.1.ntp:v3 mcast strat 3  [ttl 1]
07:44:47 172.25.1.5.ntp > 224.0.1.1.ntp:v3 mcast strat 3  [ttl 1]
07:45:51 172.25.1.5.ntp > 224.0.1.1.ntp:v3 mcast strat 3  [ttl 1]
```

The NTP server periodically forwards NTP multicast messages, while the clients on the local wire do not forward a single packet while in broadcast or multicast mode (after the initial setup). This effectively means the router can just send one packet every 64 seconds and synchronize a large number of clients.

The packet trace also displays some useful information about the server. First, the server's IP address is `172.25.1.5` and it is configured to send multicast NTP packets with the well-known NTP multicast address `224.0.1.1`. It also shows that the server is running NTP Version 3 and is advertising itself as a Stratum 3 NTP server. Finally, it shows that the NTP packets have a TTL value of 1, which will contain these NTP packets to the local LAN segment.

Since multicast traffic is more efficient than broadcast traffic, it is the preferred method of providing

NTP service via the local LAN. However, since not all NTP clients currently support NTP multicasting, you may have to also use NTP broadcast mode until all clients support multicasting. NTP broadcast services can safely coexist on the same wire as NTP multicast traffic, which should assist network administrators who are converting client software.

For redundancy purposes, you can configure multiple NTP broadcast/multicast servers on a single subnet.

14.10.4 See Also

Recipe 14.9; Recipe 14.12; Chapter 23

Top

Recipe 14.11 Enabling and Disabling NTP Per Interface

14.11.1 Problem

You want to control NTP services on a per-interface basis.

14.11.2 Solution

Depending on the level of access control required, you can use the *ntp disable* command to prevent the router from providing NTP services on a particular interface:

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#interface Serial0/1
Router(config-if)#ntp disable
Router(config-if)#end
Router#
```

You can also prevent the router from providing NTP services on an individual interface with access control lists:

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#access-list 107 deny udp any eq 123 any eq 123
Router(config)#access-list 107 permit ip any any
Router(config)#interface Serial0/1
Router(config-if)#ip access-group 107 in
Router(config-if)#end
Router#
```

Although both examples effectively disable the router from providing NTP services through the interface `Serial0/1`, the inbound access list provides more flexibility.

14.11.3 Discussion

By default, a Cisco router that has NTP services enabled automatically becomes an NTP server and provides time services to all interfaces. However, you may want to disable NTP services on one or more of the router's interfaces. For instance, you may want to prevent your router from providing NTP services to devices outside of your organization. You could accomplish this by disabling NTP on router interfaces that connect to these external networks. Further, some organizations insist that end devices

should get their NTP services from a dedicated NTP server. In this case, you might want to prohibit routers from providing time services, although they would still take part in NTP for synchronizing their own clocks.

The *ntp disable* command in the previous example prevents any NTP associations from using the `Serial0/1` interface. This affects both inbound and outbound associations. However, it will not prevent the router from routing NTP traffic through this interface on its way to another NTP device. By contrast, the access list example prevents the router from passing any NTP packets received by this interface, regardless of the destination. It will also prevent the router itself from using this interface for forming any NTP associations.

14.11.4 See Also

Recipe 14.14

Top

Recipe 14.12 NTP Authentication

14.12.1 Problem

You want to authenticate your NTP packets.

14.12.2 Solution

Use the *ntp authentication* command to authenticate NTP traffic between associations. To configure an NTP-enabled router to require authentication when other devices connect to it, use the following commands:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#ntp authentication-key 2 md5 oreilly
Router1(config)#ntp authenticate
Router1(config)#ntp trusted-key 2
Router1(config)#end
Router1#
```

Then configure the same authentication key on the client router:

```
Router2#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router2(config)#ntp authentication-key 2 md5 oreilly
Router2(config)#ntp authenticate
Router2(config)#ntp trusted-key 2
Router2(config)#ntp server 172.25.1.5 key 2
Router2(config)#end
Router2#
```

14.12.3 Discussion

People often confuse authentication with encryption. Authentication proves the authenticity of a packet's source, whereas encryption encodes or enciphers the packet contents. For the purposes of NTP, proving the authenticity of the packet is critical, but encrypting the contents of the packet is unnecessary, since it only contains time information (which isn't terribly sensitive).

Cisco fully supports NTP authentication as defined in RFC 1305. Authentication ensures that NTP associations synchronize time only to known and trusted NTP servers. This prevents servers from masquerading as legitimate timeservers, either accidentally or intentionally.

Time services and the ability to manipulate time services are critical to organizations that may depend on accurate time for billing, business regulatory, fault isolation, or security purposes. In such organizations, there is at least a theoretical danger that somebody could put a false NTP server on the network to change the clocks, which could be useful in some larger nefarious scheme. To prevent such problems and enhance security, many organizations allow authenticated NTP relationships only within their corporate networks.

The NTP protocol uses the RSA Message Digest 5 (MD5) algorithm to provide cryptographic authentication of NTP packets. Although the NTP packet is not encrypted, a one-way hash, created using pre-shared keys, ensures the authenticity of the sender and packet.

The previous example shows a single client/server pair enabling authentication between them. The NTP client and server pair must share the same key number and value ("oreilly") before authentication will work. It is important to note that routers support multiple keys and can assign a different key for each association, if required. You can also configure the router to accept time updates from a mixture of authenticated and nonauthenticated servers. The example below shows portions of an NTP debug trace:

```
Router1#debug ntp packet
NTP packets debugging is on
Router1#debug ntp authentication
NTP authentication debugging is on
Mar 18 22:39:12 EST: NTP: rcv packet from 172.16.2.2 to 172.25.1.7
Mar 18 22:39:12 EST:  leap 0, mode 3, version 3, stratum 4, ppoll 256
Mar 18 22:39:12 EST:  Authentication key 2
Mar 18 22:39:12 EST: NTP: stateless xmit packet to 172.16.2.2:
Mar 18 22:39:12 EST:  leap 0, mode 4, version 3, stratum 3, ppoll 256
Mar 18 22:39:12 EST:  Authentication key 2
```

Notice that the client polled the server with authentication key 2 and the server responded with its NTP response and authentication key 2 as well. Assuming that both key 2 strings are also equal, the client will form an association with the server and synchronize its internal clock with the server's.

Broadcast and multicast NTP associations also support NTP authentication:

```
Router3#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router3(config)#clock timezone EST -5
Router3(config)#clock summer-time EDT recurring
Router3(config)#ntp authentication-key 2 md5 oreilly
Router3(config)#ntp trusted-key 2
Router3(config)#ntp server 172.25.1.1
Router3(config)#interface ethernet0
Router3(config-if)#ntp multicast key 2
Router3(config-if)#end
Router3#
```

This example shows the configuration for an NTP multicast server that sends multicast packets using the default NTP address, `224.0.1.1`, with authentication enabled. You can configure authentication for NTP broadcast mode similarly by adding the command:

```
Router3(config-if)#ntp broadcast key 2
```

It is important to note that NTP broadcast and multicast modes use a single key for the entire multicast domain.

Organizations that receive NTP feeds from the Internet cannot usually rely on NTP authentication because few public servers support authentication functionality. This is not a major concern since NTP algorithms ignore timeservers with outrageous dates and times. Reduce the risk of synchronizing to illegitimate NTP providers by configuring your NTP servers with multiple NTP Stratum 1 servers.

Top

Recipe 14.13 Limiting the Number of Peers

14.13.1 Problem

You want to limit the number of NTP peers that the router will accept.

14.13.2 Solution

Use the *ntp max-associations* configuration command to limit the number of NTP associations the router will accept:

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#ntp max-associations 30
Router(config)#end
Router#
```

14.13.3 Discussion

To prevent NTP associations from using too many valuable resources, Cisco provides the ability to limit the number of associations that a router will accept. While the *ntp max-associations* command limits the number of inbound NTP associations, it does so without prejudice. The recipe example permits the first 30 NTP associations, regardless of where they came from.

Other methods of controlling NTP associations mentioned in this chapter provide greater control and granularity than just limiting the number.

14.13.4 See Also

Recipe 14.14

Top

Recipe 14.14 Restricting Peers

14.14.1 Problem

You want to restrict your router's NTP services.

14.14.2 Solution

You can use the *ntp access-group* command to restrict which devices you want your router to allow NTP associations with:

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#access-list 88 permit host 172.25.1.1
Router(config)#access-list 88 permit host 10.1.1.1
Router(config)#access-list 99 permit 172.25.0.0 0.0.255.255
Router(config)#access-list 99 permit 10.2.0.0 0.0.255.255
Router(config)#clock timezone EST -5
Router(config)#clock summer-time EDT recurring
Router(config)#ntp server 172.25.1.1 version 3
Router(config)#ntp server 10.1.1.1 version 3
Router(config)#ntp access-group peer 88
Router(config)#ntp access-group serve-only 99
Router(config)#end
Router#
```

14.14.3 Discussion

In this example, the router will allow the internal clock to be synchronized by the two NTP servers listed in access list 88,`172.25.1.1` and `10.1.1.1`. The router also allows time requests only from the client devices permitted by access list 99.

By default, NTP has no access controls, and it gives full access to all NTP devices. The *ntp access-group* command limits this access to various NTP services. In the example above, the *peer* keyword means that the router will only allow its internal clock to be changed by those remote servers and peers permitted by the access list.

The *serve-only* keyword specifies the clients permitted to obtain time services from the router. In the above example, the serve-only access list (99) permits two entire subnets, `172.2.0.0 255.255.0.0` and `10.2.0.0 255.255.0.0`. This means that any NTP clients residing on either of these two subnets can

obtain time services from the local router. Using the same method, you can limit the access list to a single subnet, a group of hosts, or no one. Omitting the *ntp access-group serve-only* command completely prevents the router from providing time services.

NTP access groups provide excellent granularity of access to time services on a global basis. Used in conjunction with the interface command *ntp disable*, NTP access groups can form the basis of an effective access control strategy.

14.14.4 See Also

Recipe 14.11

Top

Recipe 14.15 Setting the Clock Period

14.15.1 Problem

You want to improve on the default value in the *ntp clock-period xxxxxx* command that automatically appears when you configure NTP on your router.

14.15.2 Solution

The router will automatically generate an *ntp clock-period* line that it uses to help speed resynchronization after a reboot. Leave it alone:

```
Router#show running-config | include clock-period
ntp clock-period 17180200
Router#
```

Do not attempt to remove or modify the *ntp clock-period* command. The router automatically generates this command to compensate for internal timer inaccuracies.

14.15.3 Discussion

Do not modify or remove this statement from the router configuration. The router automatically adds this statement to the configuration when you enable NTP. After initializing NTP, the router tunes the clock period to compensate for precision problems with the router's internal clock.

The router will actually change the NTP clock period as it improves its estimation of how fast or slow its clock runs. For this reason, Cisco suggests that all routers have their configurations saved to NVRAM one week after enabling NTP. This ensures that the saved clock period is accurate for the next router initialization and allows for quicker NTP synchronization.

Top

Recipe 14.16 Checking the NTP Status

14.16.1 Problem

You want to verify the status of NTP on your router to make sure it's running properly.

14.16.2 Solution

Use the NTP and clock *show* commands to verify the status of NTP on your router. The best place to start is the *show clock detail* command, which provides information on the current time, time source, and time zone configuration:

```
Router>show clock detail
14:05:04.299 EST Mon Mar 10 2003
Time source is user configuration
Summer time starts 02:00:00 EST Sun Apr 6 2003
Summer time ends 02:00:00 EDT Sun Oct 26 2003
Router>
```

To display the current NTP status of the local router, the command is:

```
Router>show ntp status
```

The command to display the current NTP associations is:

```
Router>show ntp associations
```

You can display detailed information about the current NTP associations with the following command:

```
Router>show ntp associations detail
```

14.16.3 Discussion

You can view the current clock status using the *show clock detail* command:

```
Router>show clock detail
.23:13:41.415 EST Sun Mar 16 2003
Time source is NTP
Summer time starts 02:00:00 EST Sun Apr 6 2003
Summer time ends 02:00:00 EDT Sun Oct 26 2003
Router>
```

In this example, the router's current date and time are correctly set, but notice the period in front of the time (some routers display an asterisk instead of a period). This indicates that NTP is unsynchronized. The router will put this indicator in the output of show commands and in log messages. This allows you to spot problems with NTP even when you are not specifically looking for timing issues.

This command also provides valuable information on time zone and Daylight Saving Time. The router in the previous example is set to Eastern Standard Time (EST) and observes Daylight Saving Time.

To display detailed NTP status information, use the following *show* command:

```
Router>show ntp status
Clock is synchronized, stratum 4, reference is 172.16.2.1
nominal freq is 250.0000 Hz, actual freq is 249.9976 Hz, precision is 2**18
reference time is C03AA3D1.64A71D43 (23:21:37.393 EST Wed Mar 13 2003)
clock offset is -0.6743 msec, root delay is 99.95 msec
root dispersion is 1940.67 msec, peer dispersion is 1876.65 msec
Router>
```

In this example, the router is synchronized to NTP server `172.16.2.1` and is acting as a Stratum 4 NTP server. This means that this router is four NTP "hops" away from the authoritative clock and that it is using the server `172.16.2.1` as its master time source. Any clock status other than "synchronized" indicates problems.

To display the status of all NTP associations, use the following command:

```
Router>show ntp associations
      address         ref clock     st  when  poll reach  delay  offset    disp
+~172.25.1.1       192.5.41.40      2   148  1024  377    30.0   33.66     8.0
*~172.25.1.3       192.5.41.40      2    42  1024  377    31.3  -69.53    10.8
+~172.25.1.5       172.25.1.3       3   780  1024  377    31.9  -107.7    26.1
* master (synced), # master (unsynced), + selected, - candidate, ~ configured
Router>
```

This command gives a table of outbound NTP associations and their current status, but it does not display the current inbound NTP associations that are receiving time from this router. It shows only those NTP associations that have been explicitly configured, as well as any broadcast and multicast servers that have been discovered.

There is a lot of useful information in this example. The most important is the NTP synchronization status for each destination. The special characters on the far left indicate the current status of the NTP associations. Table 14-2 shows what these symbols mean.

Table 14-2. NTP peer status codes

Character	Description
*	Synchronized to master
#	Close to synchronization
+	Selected for possible synchronization
-	Candidate for synchronization
~	Statically configured

The previous example displays three synchronized NTP associations, with one of them (`172.25.1.3`) acting as master. The table also shows the reference clock for each neighbor, the NTP server they are synchronized to, and the current stratum number of each. Finally, the table displays polling information, availability statistics, and clock offset timing. You can view detailed NTP association information by adding the *detail* keyword:

```
Router>show ntp associations detail
172.25.1.1 configured, selected, sane, valid, stratum 2
ref ID 192.5.41.40, time C034F56D.49F8D2E5 (20:56:13.288 UTC Sat Mar 9 2003)
our mode client, peer mode server, our poll intvl 1024, peer poll intvl 1024
root delay 113.69 msec, root disp 67.46, reach 377, sync dist 148.895
delay 29.95 msec, offset 33.6585 msec, dispersion 8.00
precision 2**20, version 3
org time C034F86F.1132617C (21:09:03.067 UTC Sat Mar 9 2003)
rcv time C034F86F.0C6A77E0 (21:09:03.048 UTC Sat Mar 9 2003)
xmt time C034F86F.04B86272 (21:09:03.018 UTC Sat Mar 9 2003)
filtdelay =    29.95   29.77    4.24    5.19    5.20    5.08    5.22    5.26
filtoffset =   33.66   35.53   49.12   46.62   46.97   47.24   47.56   45.92
filterror =     0.02   15.64   31.27   46.91   62.53   78.16   93.78  109.41

172.25.1.3 configured, our_master, sane, valid, stratum 2
ref ID 192.5.41.40, time C034F647.75CB252C (20:59:51.460 UTC Sat Mar 9 2003)
our mode client, peer mode server, our poll intvl 1024, peer poll intvl 1024
root delay 135.13 msec, root disp 51.15, reach 377, sync dist 145.172
delay 31.30 msec, offset -69.5341 msec, dispersion 10.82
precision 2**20, version 3
org time C034F8D8.F76E503F (21:10:48.966 UTC Sat Mar 9 2003)
rcv time C034F8D9.0D3CCF79 (21:10:49.051 UTC Sat Mar 9 2003)
xmt time C034F8D9.0531A98E (21:10:49.020 UTC Sat Mar 9 2003)
filtdelay =    31.30   29.75    6.39    5.40    5.34    5.37    5.33    5.34
filtoffset =  -69.53  -66.27  -51.98  -50.29  -50.13  -49.93  -46.16  -43.89
filterror =     0.02   15.64   31.27   46.91   62.53   78.16   93.78  109.41
Router>
```

This command displays much of the same information, but in greater detail. It also includes several pieces of timer information. The first line of each association is of particular interest. For example, server `172.25.1.3` is currently acting as this router's master, meaning that we are synchronized to it, and that it is acting as a Stratum 2 server with a valid and sane time.

Top

Recipe 14.17 Debugging NTP

14.17.1 Problem

You want to debug and isolate NTP problems.

14.17.2 Solution

Use the *show ntp association* command to view the status of the configured NTP associations:

```
Router>show ntp associations
```

Use the *ping* command to ensure connectivity to the NTP server exists:

```
Router>ping 172.25.1.1
```

Use the *debug ntp packet* command to view the NTP packets being generated by the router:

```
Router#debug ntp packets
NTP packets debugging is on
```

14.17.3 Discussion

If the router's internal clock is incorrect and the router has NTP enabled, then the first step is check the status of the NTP associations:

```
Router>show ntp associations
      address         ref clock     st  when  poll reach  delay  offset    disp
   ~172.25.1.5       0.0.0.0        16     -    64    0     0.0    0.00  16000.
   ~10.1.1.1         192.168.15.32   2    60    64    0    27.6  -1100.  16000.
 * master (synced), # master (unsynced), + selected, - candidate, ~ configured
Router>
```

Note that there are two NTP associations configured on this router but neither is currently our synchronized master. The example also indicates that neither of the two NTP associations are currently reachable since the "reach" statistic is zero.

The most obvious place to begin is to test connectivity. You can test connectivity from the router to its NTP association with the *ping* command. If the NTP association does not respond to the *ping* request, a network path may be obstructed or the peer may be down. Isolating and fixing the connectivity issues may rectify the NTP problem. Note that access control lists between the router and its NTP peer may

prevent *ping* traffic from passing, but allow NTP (or vice versa):

```
Router>ping 172.25.1.5
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 172.25.1.5, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 1/2/4 ms
Router>
```

This shows that at least one of the configured NTP associations is reachable, so the problem might be NTP-specific, not a connectivity issue. Recall that earlier in the chapter we mentioned that an NTP client never synchronizes its clock to an unsynchronized server. To ensure that the NTP server is stable, you can either log in to the server device to check it, or, if you do not have administrative access, you can use the router's debug facility:

```
Router#debug ntp packet
NTP packets debugging is on
.Mar 21 02:39:18: NTP: xmit packet to 172.25.1.5:
.Mar 21 02:39:18:  leap 3, mode 3, version 3,  stratum 0, ppoll 64
.Mar 21 02:39:18:  rtdel 28C7 (159.286), rtdsp 2444 (141.663), refid AC190101
.Mar 21 02:39:18:  ref C043C43F.47A9CD5C (21:30:23.279 EST Wed Mar 20 2003)
.Mar 21 02:39:18:  org 00000000.00000000 (19:00:00.000 EST Thu Dec 31 1899)
.Mar 21 02:39:18:  rec 00000000.00000000 (19:00:00.000 EST Thu Dec 31 1899)
.Mar 21 02:39:18:  xmt C043C656.4DFC7394 (21:39:18.304 EST Wed Mar 20 2003)
.Mar 21 02:39:25: NTP: rcv packet from 172.25.1.5 to 172.16.2.2 on Fa0/0.1:
.Mar 21 02:39:25:  leap 3, mode 3, version 3,  stratum 0, ppoll 64
.Mar 21 02:39:25:  rtdel 286E (157.928), rtdsp 0EC6 (57.709), refid AC190101
.Mar 21 02:39:25:  ref C043C4D7.1D633CDE (21:32:55.114 EST Wed Mar 20 2003)
.Mar 21 02:39:25:  org 00000000.00000000 (19:00:00.000 EST Thu Dec 31 1899)
.Mar 21 02:39:25:  rec 00000000.00000000 (19:00:00.000 EST Thu Dec 31 1899)
.Mar 21 02:39:25:  xmt C043C65D.1D0A6CBC (21:39:25.113 EST Wed Mar 20 2003)
.Mar 21 02:39:25:  inp C043C65D.1296E3C7 (21:39:25.072 EST Wed Mar 20 2003)
```

This output shows a debug trace of NTP packets. The first packet captured is an active NTP poll from the local router to its NTP server, `172.25.1.5`. The second packet captured is the passive response from the NTP server and its current stratum level is set to zero. A zero Stratum level reported by the server means that it is currently not synchronized. If the server is not synchronized, our router can't use it. The process of debugging the NTP problem can now shift to the NTP server.

NTP is remarkably stable. Typically, when there are problems, they are related to connectivity: a box has failed, a path has become unreachable, or an access list has prevented NTP from communicating. As we have discussed, upstream problems affect downstream NTP associations, since nothing will synchronize to an unsynchronized server. This means that NTP issues must be isolated one hop at a time.

Top

Chapter 15. DLSw

Top

Introduction

There are essentially two kinds of bridges. The first type is a Source Route Bridge (SRB), which allows end devices to request a particular path through the network using a Routing Information Field (RIF) in the packet. In the default case, this type of bridge cannot forward any packet without a RIF. The second type is a Transparent Bridge, which hides all of that network detail from end devices. Transparent Bridges have no concept of a RIF. SRBs are commonly used with Token Ring networks, while Transparent Bridging is popular with Ethernets, where it is used by Ethernet switches.

Bridging between Ethernet and Token Ring networks requires a special hybrid of these two that is able to translate between not only the media types, but also the bridging types. The Remote Source Route Bridging (RSRB) and Source Route Transparent (SRT) bridging protocols were invented to solve this problem, particularly over WANs.

Data Link Switching (DLSw) and DLSw+, which is Cisco's enhanced version of DLSw, also solve these problems and comply with the same bridging standards. These protocols are capable of connecting Token Rings to Ethernets, Synchronous Data Link Control (SDLC) serial connections, and even X.25 networks. So there is really very little reason to worry about the older bridging protocols and methods anymore. If you are considering building a new network involving the System Network Architecture (SNA) protocol, there is no particular reason to bother with either SRB or RSRB. If you have an existing network involving these protocols, it would be wise to consider moving to the more modern and flexible DLSw or DLSw+.

Because DLSw creates bridges that are able to connect different (or similar) Layer 2 media together, it clearly has many applications beyond SNA, although that is the most common reason for deploying DLSw. It can also be used when bridging LAN segments for other non-routable protocols such as NetBIOS and Local Area Transport (LAT). And it can be used in conjunction with routing on the same interfaces so that some protocols are routed and others are bridged.

DLSw is an open standard protocol for bridging through TCP/IP networks. It was originally developed by IBM as a proprietary standard in 1992 and became an open standard with the publication of RFC 1434 the following year. Version 1 of the DLSw protocol was defined in detail in 1995 in RFC 1795, and updated to create Version 2 in 1997 in RFC 2166. This set of updates does not affect the underlying protocol, but rather extends its functionality. Meanwhile, Cisco independently implemented a distinct set of extensions to DLSw Version 1 and called the result DLSw+.

There are currently three different common versions of the protocol with different capabilities supported by different vendors: Version 1, Version 2, and DLSw+. Fortunately, all versions include a capabilities field that is used when two devices first attempt to make a DLSw connection. This allows them to

agree on a set of common features. In most cases this results in good transparency of operation among different vendors. However, it is useful to be aware of what features will not be supported when interconnecting in this way.

Most of the DLSw+ enhancements allow for greater scalability and variety of transport mechanisms. For example, DLSw+ allows the transport mechanism to be Fast-Sequenced Transport (FST), Frame Relay, or High-Level Data Link Control (HDLC) protocols (as well as TCP/IP). This book covers only the TCP/IP version, however. Other DLSw+ enhancements, such as peer groups and border peers, improve scalability and allow you to build a large bridged network out of smaller groups of devices that pass limited amounts of information between them as required.

Service Access Points (SAP and LSAP)

The Logical Link Control layer, IEEE 802.2, defines Service Access Points (SAP) and Link Service Access Points (LSAP). These are conceptually similar to TCP port numbers in many ways, although it is important to remember that they operate at the Logical Link Layer (Layer 2), not the Transport Layer (Layer 4), as TCP does. They are simply numbers that a device uses when it wants to establish a connection to another device to run a particular application. The number specifies a particular application protocol. The packets establishing a connection specify both a Source SAP number (SSAP) and a Destination SAP number (DSAP). These are, obviously enough, the SAP numbers of the source and destination applications. Table 15-1 lists several of the most common SAP numbers.

Table 15-1. Common SAP numbers

Hex SAP number	Binary SAP number	Description
00	0000 0000	Null LSAP
02	0000 0010	Individual LLC sublayer management
03	0000 0011	Group LLC sublayer management
04	0000 0100	Individual SNA path control
05	0000 0101	Group SNA path control
06	0000 0110	IP
07	0000 0111	IP
08	0000 1000	SNA
09	0000 1001	SNA

Hex SAP number	Binary SAP number	Description
0C	0000 1100	SNA
0D	0000 1101	SNA
0E	0000 1110	PROWAY network management and initialization
18	0001 1000	Texas Instruments
42	0100 0010	802.1 spanning tree protocol
4E	0100 1110	EIA RS-511 manufacturing message service
7E	0111 1110	X.25 over 802.2 LLC
80	1000 0000	Xerox Network Systems (XNS)
86	1000 0110	Nestar
8E	1000 1110	PROWAY active station list maintenance
98	1001 1000	Address Resolution Protocol (ARP)
AA	1010 1010	Sub-Network Access Protocol (SNAP)
BC	1011 1100	Banyan Vines
E0	1110 0000	Netware
F0	1111 0000	NetBIOS
F4	1111 0100	Individual LAN management
F5	1111 0101	Group LAN management
F8	1111 1000	Remote Program Load (RPL)
FA	1111 1010	Ungermann-Bass
FE	1111 1110	ISO network layer protocol
FF	11111111	Global LSAP

Cisco routers include the ability to filter based on LSAP numbers using access lists in the range from 200 to 299. Here is an example of the syntax of an LSAP access list:

```
access-list 201 permit 0x0000 0x0D0D
```

The first hexadecimal number after the *permit* keyword represents both SSAP and DSAP. The first two hex digits are the SSAP, and the second two are the DSAP. The next field is a wildcard bit pattern. When the wildcard has a 0 bit, the corresponding bit in the SAP numbers must be exactly as it is in the given

pattern, and any place where the wildcard has a 1 bit can have either a zero or a one.

The mask in this particular example is `0x0D0D`. The hex number `D` has a bit pattern of `1101`. So, the access list as written will allow any packets with either SSAP or DSAP values shown in Table 15-2.

Table 15-2. SAP values matched by the example ACL

Hex	Binary	SAP
0x00	0000 0000	Null LSAP
0x01	0000 0001	Unused
0x04	0000 0100	Individual SNA path control
0x05	0000 0101	Group SNA path control
0x08	0000 1000	SNA
0x09	0000 1001	SNA
0x0D	0000 1101	SNA

Such access lists are usually used to block unwanted local ring traffic such as NetBIOS or Netware, while permitting the SNA traffic. If, on the other hand, you wanted to permit only NetBIOS traffic and block all other protocols, you could use an access list like this:

```
access-list 202 permit 0xF0F0 0x0000
```

Explorers and RIFs

When a device wants to send a packet using Logical Link Control (LLC) protocols through a bridged network, it has the capability of *source-routing* this packet. This means that the end device is able to specify a particular network path. To do this, however, it first has to find an appropriate path. It does this by sending a packet called an *explorer* through the network. As this explorer packet passes through the network, each bridge adds information about itself to the packet and forwards it along. So when it finally arrives, it has a complete path description that the end device can use to build a RIF.

There are, in fact, two different kinds of explorers, called *spanning tree explorers* and *all routes explorers*. They both perform the same basic function of trying to map the best path to the required destination. The difference, however, is that a spanning tree explorer follows only one path, and the all routes explorer attempts all paths. When a bridge receives a spanning tree explorer, it forwards the packet along a single path defined by the Spanning Tree Protocol (STP).

STP eliminates loops from a bridged network. It is important to remember that running STP is optional,

and not every bridge is configured to run it. It is not frequently used in DLSw+ networks because the protocol has the ability to do useful things such as load sharing between links. STP inherently prevents load sharing among the many different possible paths through a network by shutting down all paths except for one.

Note that STP is required on transparently bridged networks, however, because there is no RIF to control path selection. If you have multiple connections between transparent bridges, such as Ethernet switches, you must use STP.

STP ensures that there is one and only one active path between any two points by first electing a *root bridge*. This device is the logical center of the bridged network. When a bridge receives a packet destined for a device that is not on one of its ports, it simply forwards that packet toward the center. The packet may take several hops to reach the root bridge, which has an exhaustive table of MAC addresses and knows how to forward every packet that it receives. If it doesn't know the destination, it will duplicate the packet and send it out every path except the one it was received on, in the hopes of finding the destination.

A spanning tree explorer packet simply follows the STP path through the network to reach the required destination. An all routes explorer works similarly, but it follows all possible paths to reach the ultimate destination. At each bridge where there is a choice to be made between two or more possible paths, the bridge duplicates the packet and forwards it along all of them. So the destination device will probably receive several possible solutions. In general, it will pick the first one it receives on the assumption that this must represent the fastest path.

When the destination device receives an explorer packet, it turns it around and sends it back to the original source, retaining the routing information. Now both devices know how to request a path to one another through the network when they need to exchange information. When they send packets of application data, they will include a Routing Information Field (RIF) that specifies the desired path.

This process can obviously get messy if there are a lot of devices all trying to find one another at the same time. So DLSw+ includes some optimizations that allow routers to improve on the RIF discovery process. Every router contains a RIF cache of all of the remote devices that it knows how to reach. When a device on the local Token Ring sends an explorer looking for something the router already knows how to reach, DLSw doesn't need to bother forwarding this explorer through the network. Instead, it responds directly without having to consume network resources forwarding the explorer.

Cisco IOS Code Sets

One common misunderstanding that people have about DLSw+ is that to implement Cisco routers in a network using IBM's Advanced Peer-to-Peer Networking (APPN) functionality, you have to use one of Cisco's APPN code sets. This is not the case. The core DLSw+ functionality is included in the default minimal IP-Only IOS code set for all 12.x and most 11.x IOS levels. You need to use the APPN code set

only if you intend for the router to take an active part in the higher layer protocols.

APPN is effectively the next generation version of SNA. Among many improvements, it makes the protocol routable for improved scalability. However, APPN still runs over the same lower layer protocols such as LLC2 on Token Rings and SDLC on serial interfaces. So, in most cases the router doesn't need to know whether APPN or SNA is used at higher layers.

The APPN code set is required only if the router needs to provide native APPN routing. In most cases, even networks using APPN within the mainframe and its Front End Processor (FEP), the bridging functions of DLSw+ are sufficient to provide all of the required connectivity.

The most recent generations of mainframe computers from IBM are capable of supporting TCP/IP and Gigabit Ethernet directly, so we expect that the future of mainframe networking will use IP rather than APPN. In this case, SNA and DLSw will be necessary only to support legacy SNA equipment.

Top

Recipe 15.1 Configuring DLSw

15.1.1 Problem

You want to set up DLSw to allow Token Ring bridging through an IP network.

15.1.2 Solution

There are many different ways to configure two routers to allow Token Ring-to-Token Ring bridging through DLSw. The most common reason for doing this is to allow Token Ring SNA LLC2 devices to communicate with a mainframe FEP attached to another Token Ring. It is relatively common to have many remote rings connecting to a single central ring. In cases like this, it is often best to use one or more dedicated DLSw routers at the central location. The CPU overhead required for supporting a large number of DLSw connections can be relatively high, so it is useful to restrict this functionality to special-purpose DLSw routers and keep it off of routers that also need to handle core routing functions.

Here is the DLSw configuration for a central router, which is the one that connects directly to the ring that holds the FEP:

```
dlsw-central#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
dlsw-central(config)#interface Loopback0
dlsw-central(config-if)#ip address 10.1.1.5 255.255.255.252
dlsw-central(config-if)#exit
dlsw-central(config)#access-list 701 permit 4000.3745.AAAA 8000.0000.0000
dlsw-central(config)#source-bridge ring-group 101
dlsw-central(config)#dlsw local-peer peer-id 10.1.1.5 promiscuous
dlsw-central(config)#dlsw timer explorer-wait-time 5
dlsw-central(config)#dlsw icanreach mac-exclusive
dlsw-central(config)#dlsw icanreach mac-address 4000.3745.AAAA mask ffff.ffff.ffff
dlsw-central(config)#dlsw cache-ignore-netbios-datagram
dlsw-central(config)#dlsw allroute-sna
dlsw-central(config)#interface TokenRing0
dlsw-central(config-if)#description Ring number 0x00A (10)
dlsw-central(config-if)#source-bridge 10 5 101
dlsw-central(config-if)#source-bridge spanning
dlsw-central(config-if)#source-bridge input-address-list 701
dlsw-central(config-if)#end
dlsw-central#
```

And the remote routers are configured like this:

```
dlsw-branch#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
dlsw-branch(config)#interface Loopback0
dlsw-branch(config-if)#ip address 10.1.2.5 255.255.255.252
dlsw-branch(config-if)#exit
dlsw-branch(config)#access-list 200 permit 0x0000 0x0D0D
dlsw-branch(config)#source-bridge ring-group 101
dlsw-branch(config)#dlsw local-peer peer-id 10.1.2.5
dlsw-branch(config)#dlsw timer explorer-wait-time 5
dlsw-branch(config)#dlsw remote-peer 0 tcp 10.1.1.5 lsap-output-list 200
dlsw-branch(config)#dlsw allroute-sna
dlsw-branch(config)#interface TokenRing0
dlsw-branch(config-if)#description branch Token Ring 0x047 (71)
dlsw-branch(config-if)#multiring all
dlsw-branch(config-if)#source-bridge 71 5 101
dlsw-branch(config-if)#source-bridge spanning
dlsw-branch(config-if)#end
dlsw-branch#
```

15.1.3 Discussion

These configurations will work well for situations in which a large number of remote routers need to make DLSw connections to a central router. However, if you only need to connect two routers, you can still use the same pair of configurations. In that case, you simply need to decide which router will initiate the session. The initiating router will use the *dlsw-branch* configuration, and the receiving router will be *dlsw-central*. The initiating router will start trying to establish the connection as soon as it boots, and will continue periodically until it succeeds. This is true whether or not there is any traffic ready to be sent through this connection.

15.1.3.1 The peers

The first things to notice about these two configurations are the *dlsw local-peer* and *dlsw remote-peer* statements. These statements must match. In this case the branch router uses the address `10.1.1.5` to specify the remote peer. This address corresponds to the `Loopback0` interface on the central router. So, on the central router, it is critically important that the *local-peer* statement points to this same IP address.

The central router does not have a *dlsw remote-peer* statement. This is because the central router will not establish DLSw connections. The *promiscuous* keyword in the *local-peer* statement allows this router to simply accept incoming connections from other routers. In this example we expect to have many remote branch routers connecting to the same central router. Configuring a separate remote peer on the central router for every branch would be an onerous task, particularly if branch routers are frequently added and deleted. When building a many-to-one type bridge, it is generally preferable to configure the central router to simply accept all remote peers.

This means that the central router will accept a DLSw connection from anybody. Naturally it doesn't make sense to configure *promiscuous* on both routers because one of them has to initiate the connection. So the branch router is configured with normal point-to-point *local-peer* and *remote-peer* commands.

When configuring the local and remote DLSw peers, you can specify the IP address for any interface on the router, as long as both routers agree on which address is to be used. In these configuration files both local peers point to the respective `Loopback0` interfaces on each router. This is because the `Loopback0` interface is a purely internal software port that can never become unavailable due to an external failure. If the router is reachable at all through any path, then the `Loopback0` interface is reachable, and consequently the DLSw peer relationship will be kept alive. In the case of Token Ring to Token Ring bridging, it may make sense to bring down the peer relationship when one of the rings fails. In this case, you can configure the local peers to point to an IP address on the Token Ring interface itself. However, if either router supports more than one SNA interface, such as a second Token Ring or an SDLC serial port, then you will probably not want your SDLC traffic to fail just because of a Token Ring failure.

There are only two routers in this example, but it is relatively common to have a backup router on the central ring. Each router can have many remote peers, each configured with a separate *dlsw remote-peer* statement.

There are two other pieces of information in the *dlsw remote-peer* command on the branch router:

```
dlsw remote-peer 0 tcp 10.1.1.5 lsap-output-list 200
```

The number 0 in the third field is a list number. The value of 0 is the default, and is sufficient for most applications. If any other value is specified, it tells DLSw that you want to set up ring lists or port lists. You would do this if you wanted to bridge traffic from one ring to one remote DLSw peer and traffic from another ring to a different peer. For example, if your router has four different rings, and you only want rings 1, 2, and 4 (but not 3) to use this particular DLSw peer, then you would configure a port list as follows, arbitrarily calling this port list number 5:

```
Router(config)#dlsw port-list 5 tokenring 1 tokenring 2 tokenring 4
Router(config)#dlsw remote-peer 5 tcp 10.1.1.5
```

An alternative way of doing this would be to use the ring numbers. If, for example, the ring number associated with these Token Ring interfaces were 70, 95, and 142 (all of these ring numbers are in decimal rather than the more conventional hexadecimal notation, which would be 46, 5F, and 8E, respectively), you could accomplish the same thing with a *dlsw ring-list* command:

```
Router(config)#dlsw ring-list 5 rings 70 95 142
Router(config)#dlsw remote-peer 5 tcp 10.1.1.5
```

Note that, in general, if you are working with physical rings, there is usually less danger of confusion when using the *port-list* version of this command. It is easy to become confused by virtual ring

numbers and bridge numbers and so forth when constructing a ring list, particularly because most devices will represent these values in hexadecimal notation, while the router uses decimal.

In both versions, if there are other Token Ring interfaces that do not appear in any ring list, then they will not take part in any DLSw bridging. Alternatively, you could create more port lists or ring lists representing these other rings, and create more *dlsw remote-peer* statements for them, as follows:

```
Router(config)#dlsw port-list 5 tokenring 1 tokenring 2 tokenring 4
Router(config)#dlsw port-list 6 tokenring 3
Router(config)#dlsw remote-peer 5 tcp 10.1.1.5
Router(config)#dlsw remote-peer 6 tcp 10.1.1.9
```

These list numbers only have local significance to the router they are set on. So, if you are sending port list 5 to `10.1.1.5` and 6 to `10.1.1.9` (as in this example), there is nothing special required to complete that relationship on the remote peer routers. The other router doesn't know or care that it isn't getting all of this router's DLSw traffic.

The last parts of the *remote-peer* command on the branch router define an *lsap-output-list*, and associate it with access list 200. This is optional, but useful. To understand what it does, please refer to the discussion of SAPs in the introduction to this chapter.

15.1.3.2 Ring groups, ring numbers, and bridge numbers

Note the ring groups on both routers. Each router has several different ring and bridge numbers that define what connects to what. Nothing will work if you don't get these right.

Both routers have the global configuration statement that looks like this:

```
Router(config)#source-bridge ring-group 101
```

This specifies a virtual ring number that can be easily compared to a VLAN number. This ring group number is also a ring number, so it appears in RIFs. If the router has interfaces in multiple ring groups, you can simply include several of these configuration lines, one for each ring group that the router holds:

```
Router(config)#source-bridge ring-group 101
Router(config)#source-bridge ring-group 557
Router(config)#source-bridge ring-group 4031
```

This ring group number appears in the configuration of each interface that takes part in the bridge. For example, the branch Token Ring port configuration includes the following statement:

```
dlsw-branch(config)#interface TokenRing0
dlsw-branch(config-if)#source-bridge 71 5 101
```

The first number in the *source-bridge* interface configuration command is the local ring number, the

second number is an internal bridge number, and the last number is the destination ring number. DLSw will bridge packets received on this interface to the other Token Rings around the network that are also members of this ring group.

In this case, the local ring number is 71, which would be `0x047` in the more conventional hexadecimal notation for Token Ring numbers. This local ring number is the value that appears in RIFs generated for this ring. It is essentially a network number for this ring, similar to the TCP/IP concept of a network number. The range of possible values is from 1 to 4095, and each ring number must be globally unique. So no other ring in this ring group, anywhere in the network, can have the same number. And remember also that the ring group number is itself a ring number, so no ring can have the same number as the ring group. If you configure a local ring number like this that is in conflict with a pre-existing ring number, the router will detect the error and shut down the port to prevent further communication problems on the ring.

The bridge number is an integer value expressed as a decimal number between 1 and 15. In this case the target ring number is the virtual ring number representing the entire DLSw ring group. However, this same command could be used to simply bridge two rings on the same router:

```
Router(config)#interface Tokenring0
Router(config-if)#source-bridge 70 5 71
Router(config-if)#interface Tokenring1
Router(config-if)#source-bridge 71 5 70
Router(config-if)#interface Tokenring2
Router(config-if)#source-bridge 72 6 73
Router(config-if)#interface Tokenring3
Router(config-if)#source-bridge 73 6 72
```

The bridge number is simply a locally unique identifier that the router uses to specify how it will connect these rings. In this little example, there are four rings. The first two rings are connected together through bridge number 5 and the second pair connects via bridge number 6. In the larger DLSw example, the target ring is the ring group number. The bridge number is unique only to the router it is configured on, and is particularly useful when you want to have different rings connected to different destinations.

Another important *source-bridge* feature shown in the example is the input-a*ddress-list* keyword:

```
dlsw-central(config)#access-list 701 permit 4000.3745.AAAA 8000.0000.0000
dlsw-central(config)#interface TokenRing0
dlsw-central(config-if)#source-bridge input-address-list 701
```

This ensures that the bridge will pick up packets from the ring only if they have the source address shown in the access list. Note that this is the same MAC address that we used in the *dlsw icanreach* command:

```
dlsw-central(config)#dlsw icanreach mac-address 4000.3745.AAAA mask ffff.ffff.ffff
```

This is the MAC address of the FEP in our example. The reason for the input filter on the *source-bridge*

command is simply to ensure that the router only forwards packets that originate with the FEP. Bear in mind that DLSw potentially bridges a large number of rings together, so there could be a lot of strange and unexpected packets floating around on the FEP ring. In particular, we want to prevent our branch routers from talking directly to one another and wasting bandwidth.

15.1.3.3 Explorer options

As mentioned in the introduction to this chapter, the LLC protocol used in most Token Ring networks uses packets called explorers to build RIFs for source-routing packets. DLSw+ includes several options for improving the efficiency of this process.

The first of these options appears in this line, which is in both of the example router configurations:

```
dlsw-central(config)#dlsw allroute-sna
```

This command simply tells the router to use all routes explorers, rather than single route (or spanning tree) explorers when trying to find an SNA device. This command appears to be contradicted by the use of *source-bridge spanning* on the Token Ring interfaces of both routers, but it isn't actually a contradiction. In fact, this provides an important advantage. The *source-bridge spanning* command simply prevents end devices on the Token Ring from using all routes explorers. The routers themselves are still allowed to use all routes explorers when communicating between themselves, however. This way, the routers maintain control over network routing, while at the same time preventing end devices from wasting network resources by sending out all route explorers unnecessarily.

The second important explorer configuration command also appears on both routers:

```
dlsw-central(config)#dlsw timer explorer-wait-time 5
```

This statement tells the routers to wait for five seconds after sending an explorer, to ensure that it has received the results from all of the possible DLSw peers. This helps to ensure that it will always pick the best path. The default value if this command is not present is zero seconds, meaning that the router will always use the first path it sees. Clearly, this is only important when using all routes explorers.

At the host router we have configured two *icanreach* statements:

```
dlsw-central(config)#dlsw icanreach mac-exclusive
dlsw-central(config)#dlsw icanreach mac-address 4000.3745.AAAA mask ffff.ffff.ffff
```

This tells the router to restrict inbound traffic so that the only MAC address that a remote device can reach through DLSw is the one that we have configured (`4000.3745.AAAA`). This usually corresponds to the mainframe's FEP address. You can easily include several MAC address definitions in the same way if there are other devices to which you want to allow access.

This helps to prevent unwanted traffic from being bridged across your WAN. In particular, it ensures that DLSw will not bridge from one remote site to another.

Note that the end devices on the Token Ring only need the RIF representing the local DLSw router. DLSw is said to "terminate the RIF," meaning that it effectively creates three regions of RIF tables, from end device A to Router A, from Router A to Router Z, and from Router Z to end device Z. This is different from RSRB, for example, which passes RIF information through the network so that end devices can specify their paths. In DLSw, however, the routers take over path selection. This allows the router to respond to local RIF requests without passing them through the network. It also drastically reduces the storage required for RIF tables on all devices.

15.1.3.4 Other features

There are a couple of other small commands in these examples that allow specific functionality or improve efficiency. On the central router you will see the command *dlsw cache-ignore-netbios-datagram*. This simply tells the router to ignore NetBIOS names that it receives through DLSw. This is useful in large networks because it allows the central router to avoid having to maintain a large NetBIOS name cache. Since all connections are inbound anyway, the NetBIOS names of remote devices are never used in such a configuration, so this command allows you to ignore them.

Finally, on the branch router, we have included the line *multiring all* in the configuration of the Token Ring interface. There are actually several different options besides *all* for this command. The most common one is *multiring ip*. The *all* option allows all routable protocols such as TCP/IP and IPX to use the interface for routing, while purely bridged protocols like SNA are permitted to use it for bridging. Specifying *multiring ip* allows IP to use the interface, but not another routed protocol such as IPX.

Top

Recipe 15.2 Using DLSw to Bridge Between Ethernet and Token Ring

15.2.1 Problem

You want to set up DLSw to allow Token Ring-to-Ethernet bridging.

15.2.2 Solution

DLSw includes the capability to bridge different kinds of media. One common example of this is bridging an Ethernet segment to a Token Ring. In this example, we connect an Ethernet branch to the same central Token Ring DLSw router used in Recipe 15.1:

```
dlsw-ether-branch#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
dlsw-ether-branch(config)#interface Loopback0
dlsw-ether-branch(config-if)#ip address 10.1.3.5 255.255.255.252
dlsw-ether-branch(config-if)#exit
dlsw-ether-branch(config)#access-list 200 permit 0x0000 0x0D0D
dlsw-ether-branch(config)#source-bridge ring-group 101
dlsw-ether-branch(config)#dlsw local-peer peer-id 10.1.3.5
dlsw-ether-branch(config)#dlsw timer explorer-wait-time 5
dlsw-ether-branch(config)#dlsw remote-peer 0 tcp 10.1.1.5 lf 1470 lsap-output-list 200
dlsw-ether-branch(config)#dlsw bridge-group 1
dlsw-ether-branch(config)#dlsw transparent switch-support
dlsw-ether-branch(config)#dlsw allroute-sna
dlsw-ether-branch(config)#interface Ethernet0
dlsw-ether-branch(config-if)#description branch Ethernet
dlsw-ether-branch(config-if)#bridge-group 1
dlsw-ether-branch(config-if)#bridge 1 protocol ieee
dlsw-ether-branch(config-if)#end
dlsw-ether-branch#
```

15.2.3 Discussion

Before looking at this in detail, we need to stress that routable protocols such as TCP/IP always behave better when they are routed. So you should only use bridging between different media as in this example when there are unroutable protocols that must communicate between them. Perhaps the most common example is when LLC2 SNA devices must be accessed from an Ethernet segment.

The first difference between this example and the one shown in Recipe 15.1is in the *remote-peer* command, where we have added the flag *lf 1470* . This restricts the MTU of the bridged network to 1470 bytes so that the larger Token Ring frames cannot use the bridge. Token Ring supports much larger frames than Ethernet does. In a bridged network, there is no packet fragmentation facility (as there is in

routed networks). If large Token Ring packets cross the bridge to an Ethernet segment, they must be dropped.

Ideally, this MTU restriction should be made on the *remote-peer* commands at both ends. If your central router must support both Ethernet and Token Ring branches, then it makes sense to configure two *remote-peer* commands on the central router, one for each type of remote medium. The command that the central router uses for bridging to Ethernet segments should then have the MTU restricted in the same way with the *lf 1470* flag. If you like, you could also have a separate statement on the central router for remote Token Ring peers that can use the default value.

The next important difference between this Ethernet branch and the previous Token Ring branch is the presence of the command *dlsw transparent switch-support* . This is used in cases where the Ethernet segment that the router is connecting to the bridge is itself bridged. This would be the case if the local Ethernet segment contains an Ethernet switch. Remember from the introduction of this chapter that all Ethernet switches are transparent bridges. This command is particularly important when there are two redundant DLSw routers on an Ethernet segment. In such a case, the switch will become confused by the fact that the same downstream MAC addresses appear from both routers. If you are using Ethernet hubs, this command is unnecessary.

The interface configuration for Ethernet is completely different from the Token Ring example. There are three commands in this example that define the bridging characteristics of the Ethernet interface. The first is the *bridge-group* command found in the Ethernet interface configuration block. This command tells the router to associate this interface with the first logical bridge group in the router. This number is purely local to the router. You could build a bridge between two Ethernet interfaces on the same router as follows:

```
Router(config)#interface Ethernet0
Router(config-if)#bridge-group 1
Router(config-if)#exit
Router(config-if)#interface Ethernet1
Router(config-if)#bridge-group 1
```

But, in this example, we want to connect the Ethernet port to a Token Ring on another router. To do so, we associate this bridge group instead with DLSw using the *dlsw bridge-group* global configuration command:

```
dlsw-ether-branch(config)#dlsw bridge-group 1
dlsw-ether-branch(config)#interface Ethernet0
dlsw-ether-branch(config-if)#bridge-group 1
```

The bridging behavior is defined with the following command:

```
dlsw-ether-branch(config-if)#bridge 1 protocol ieee
```

This command tells the router to use the IEEE 802.1d STP for avoiding loops when creating the bridge instead of the older and incompatible Digital Equipment Corporation (DEC) standard. Unless you are

connecting to extremely old equipment that doesn't support the IEEE standard, you should always use this command.

15.2.3.1 Ethernet II or 802.3 framing

Bridging Ethernet frames to Token Ring frames actually introduces an important ambiguity. There are two different Ethernet framing standards, the older Ethernet II standard that is commonly used for TCP/IP, and the newer IEEE 802.3 standard that is used for many other protocols. The most immediately important difference between 802.3 and Ethernet standards is whether the header field immediately after the destination and source addresses is a length (less than 1500), as in 802.3, or a type (greater than 1500), as in Ethernet II.

The translation for 802.3 Ethernet frames is relatively clean because the Token Ring frames have a similar format, as they both share the same 802.2 header format. So this is the default shown in the previous example. However, if you want to use Ethernet II framing for the bridged protocols on the Ethernet side, you have to configure the router to understand this frame type.

Bear in mind that you can run different Ethernet frame types for different protocols running on the same Ethernet segment. For example, it is relatively common to use 802.3 framing for IPX, while IP always uses Ethernet II. There is no inherent conflict in running both of these frame types at the same time. It only becomes a problem if, for example, some of the IPX devices used one frame type and some used the other.

The Ethernet II frame type field has a value of `0x80d5` when bridging Ethernet II to Token Ring. This frame type technically refers to SNA traffic, although it applies equally to any traffic bridged from a Token Ring. To tell the router to use Ethernet II framing, you must apply this globally to all source-route bridging as follows:

```
dlsw-ether-branch(config)#source-bridge enable-80d5
```

Note that this is a global command affecting all Ethernet interfaces on the router. You cannot have some interfaces using Ethernet II, while others use 802.3 framing for bridging.

15.2.4 See Also

Recipe 15.1

Top

Recipe 15.3 Converting Ethernet and Token Ring MAC Addresses

15.3.1 Problem

You want to convert the bit ordering of MAC addresses to see how they will look after passing through an Ethernet-to-Token Ring bridge.

15.3.2 Solution

The Perl script in Example 15-1 converts Ethernet addresses to the way they will appear when connected through a bridge to a Token Ring. It also performs the reverse translation of Token Ring addresses to Ethernet, which is identical.

Example 15-1. eth-tok-mac.pl

```
#!/usr/local/bin/perl
#
# eth-tok-mac.pl -- a script to convert Ethernet to Token Ring MAC
#                   addresses when bridging with RSRB or DLSw
#
$convert[0] = "0";     $convert[1] = "8";
$convert[2] = "4";     $convert[3] = "C";
$convert[4] = "2";     $convert[5] = "A";
$convert[6] = "6";     $convert[7] = "E";
$convert[8] = "1";     $convert[9] = "9";
$convert[10] = "5";     $convert[11] = "D";
$convert[12] = "3";     $convert[13] = "B";
$convert[14] = "7";     $convert[15] = "F";

if($#ARGV != 0) {usage(   );}

$input_MAC = $ARGV[0];

# first split the incoming MAC into bytes
$_ = $input_MAC;
s/[.:-]//g;

for ($i=0; $i*2 < length($_); $i++) {
    @input_bytes[$i] = substr($_, $i*2, 2);
}

for ($i=0; $i <= $#input_bytes; $i++) {
```

```
    $_ = @input_bytes[$i];

    # first check that there aren't any illegal characters in this address
     if(/[^0-9a-fA-F]/) {
        usage(  );
    }

    if (length(  ) =  = 2 ) {
        @output_bytes[$i] = $convert[hex(substr($_, 1, 1))]
                          . $convert[hex(substr($_, 0, 1))];
    } else {
        usage(  );
    }
}

print "the resulting MAC is: ";
for ($i=0; $i < $#input_bytes; $i++) {
    print "@output_bytes[$i]-";
}
print "@output_bytes[$#input_bytes]\n";

sub usage(  ) {
  print "usage: eth-tok-mac.pl <MAC>\n";
  print "     where <MAC> is in the form HH:HH:HH:HH:HH:HH\n";
  print "     or HH-HH-HH-HH-HH-HH or HHHH.HHHH.HHHH
  print "     (H is a hex number 0-F)\n";
  print "The output is the converted MAC address.\n";
  print "Note that this conversion is exactly the same whether converting\n";
  print "from Ethernet to Token Ring or Token Ring to Ethernet.\n";

  exit;
}
```

The program is run as follows:

```
$ eth-tok-mac.pl 00-00-0c-f0-84-60
the resulting MAC is: 00-00-30-0f-21-06
```

15.3.3 Discussion

Token Ring uses a convention of most significant bit first when writing a byte. Ethernet, on the other hand, puts the least significant bit first. So, when a bridge connects these two media, the MAC addresses of devices on the Ethernet side will look unfamiliar when viewed from the Token Ring side, and vice versa. The rule for converting from one to the other is relatively simple, however, because it just reflects this reversing of the bit ordering.

Table 15-1Table 15-1 shows how the conversion algorithm works.

Table 15-3. Converting Token Ring to Ethernet MAC addresses

Token Ring			Ethernet		
Hex	Decimal	Binary	Binary	Decimal	Hex
0	0	0000	0000	0	0
1	1	0001	1000	8	8
2	2	0010	0100	4	4
3	3	0011	1100	12	C
4	4	0100	0010	2	2
5	5	0101	1010	10	A
6	6	0110	0110	6	6
7	7	0111	1110	14	E
8	8	1000	0001	1	1
9	9	1001	1001	9	9
A	10	1010	0101	5	5
B	11	1011	1101	13	D
C	12	1100	0011	3	3
D	13	1101	1011	11	B
E	14	1110	0111	7	7
F	15	1111	1111	15	F

This table converts each individual hex character in a MAC address, but there is more to it than this because a byte is 8 bits long, not 4 bits like a hex character.

Suppose the Ethernet address is `0000.0cf0.8460`. The third byte of this address is `0c`. To figure out how this byte will look on the Token Ring side of the bridge, switch the two hex characters to get `c0`. Then convert each of these characters using Table 15-1. The `c` becomes `3` and the `0` stays the same, giving a final value of `30`. Converting the entire address similarly gives `00-00-30-0f-21-06`.

The script provides an automated way to do these translations. If the 8 bits in a byte are numbered from 1-8, this algorithm simply flips the order so that they appear from 8-1. This is clearly identical to the inverse translation where the bit order is converted from 8-1 to 1-8. So both the script and the above manual technique work identically when converting Ethernet addresses to Token Ring or Token Ring

addresses to Ethernet.

15.3.4 See Also

Recipe 15.2

Top

Recipe 15.4 Configuring SDLC

15.4.1 Problem

You want to configure a serial port to connect to an SDLC device so that it can use DLSw to talk to a central mainframe.

15.4.2 Solution

The global configuration commands in this example are identical to those shown in Recipe 15.1 for using DLSw+ to connect two Token Rings. The central router's configuration is identical to what was used in Recipe 15.1, so the following shows only the remote branch configuration:

```
dlsw-branch#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
dlsw-branch(config)#interface Loopback0
dlsw-branch(config-if)#ip address 10.1.2.5 255.255.255.252
dlsw-branch(config-if)#exit
dlsw-branch(config)#access-list 200 permit 0x0000 0x0D0D
dlsw-branch(config)#source-bridge ring-group 101
dlsw-branch(config)#dlsw local-peer peer-id 10.1.2.5
dlsw-branch(config)#dlsw timer explorer-wait-time 5
dlsw-branch(config)#dlsw remote-peer 0 tcp 10.1.1.5 lsap-output-list 200
dlsw-branch(config)#dlsw allroute-sna
dlsw-branch(config)#interface Serial1
dlsw-branch(config-if)#description Connection to one remote SDLC device
dlsw-branch(config-if)#encapsulation sdlc
dlsw-branch(config-if)#no keepalive
dlsw-branch(config-if)#nrzi-encoding
dlsw-branch(config-if)#clock rate 4800
dlsw-branch(config-if)#sdlc role primary
dlsw-branch(config-if)#sdlc vmac 4000.CCCC.0000
dlsw-branch(config-if)#sdlc poll-pause-timer 200
dlsw-branch(config-if)#sdlc address 20
dlsw-branch(config-if)#sdlc xid 20 017A0006
dlsw-branch(config-if)#sdlc partner 4000.3745.AAAA 20
dlsw-branch(config-if)#sdlc dlsw 20
dlsw-branch(config-if)#end
dlsw-branch#
```

15.4.3 Discussion

SDLC is a serial protocol that is commonly used for SNA devices. It is extremely popular in financial environments for devices such as banking machines, card readers, and account printers. Using SDLC in conjunction with DLSw is becoming increasingly common in these sorts of situations. This allows the remote SDLC devices to communicate with the Token Ring Interface Coupler (TIC) on a mainframe computer through an IP network. There is a huge advantage to this because it means that you don't have to have a separate serial port on your mainframe's Front End Processor (FEP) for every remote SDLC connection. Instead, you can have a single Token Ring interface supporting hundreds of remote devices that are connected through a reliable routed TCP/IP network.

Although it is essentially a synchronous serial protocol, there are several ways of constructing SDLC links. The simplest is a point-to-point connection from the router to an SDLC device, perhaps even another router. The SDLC link can also connect several devices in a long serial bus connection called a *multidrop*, and it can even be set up in a ring topology. To accomplish this, SDLC requires that each device have a unique 8-bit address between `0x01` and `0xFF` in hexadecimal (1-255 in decimal notation). Each device is configured to respond to one of these SDLC addresses. A primary SDLC device polls every known address in turn to ask the devices if they have data to send. So, despite having many physical devices sharing a serial connection, there is no danger of collisions from two devices sending simultaneously. One of the advantages to DLSw is that it takes care of local acknowledgement of these SDLC polls. So if you connect two SDLC circuits together using DLSw, the polling traffic doesn't cross the WAN, which saves bandwidth.

The example sets up the SDLC encapsulation on the port, provides the Data Communications Equipment (DCE) clock, and acts as the master device to a downstream SDLC device. The example also defines an artificial Token Ring MAC address for this device so that the FEP will be able to communicate with it through its Token Ring port. Let's look at these functions separately:

```
dlsw-branch(config)#interface Serial1
dlsw-branch(config-if)#encapsulation sdlc
dlsw-branch(config-if)#nrzi-encoding
dlsw-branch(config-if)#clock rate 4800
```

First, the command *encapsulation sdlc* sets up the interface to use SDLC framing. This is accompanied by the *nrzi-encoding* command that specifies the electrical characteristics of the signaling. Non-Return to Zero Inverted (NRZI) is often required with IBM equipment. The default behavior for SDLC connections on Cisco routers is the similarly named but very different Non-Return to Zero (NRZ) encoding. You can configure the router to use NRZ by just turning off NRZI encoding with the command *no nrzi-encoding*.

Next, this example assumes that the router is the DCE for this SDLC line, so it must supply the clock, with the *clock rate* command. In the example, the clock rate is set to 4800bps. In general, you want the clock rate to be as high as the downstream devices can support, to facilitate better data transfer rates. However, it is relatively common to see SDLC devices that work only intermittently at higher rates such as 19,200bps. So a good practice when having problems with SDLC equipment is to turn the clock rate down somewhat and see if the problems go away.

Note that any time that you configure a router serial interface to be the DCE device, you must use a DCE cable. The router will not even accept the *clock rate* command if you have a DTE cable connected to the interface.

The command *sdlc role primary* is used with multidrop SDLC connections to define the router's function in the multidrop network:

```
dlsw-branch(config-if)#sdlc role primary
```

A multidrop connection simply means that several devices are connected in series to the same serial port. You lose nothing by using this command with a single downstream device connected directly to the router, however. The different options for the *sdlc role* command are *primary*, *secondary*, and *prim-xid-poll*. Configuring the router to be the SDLC primary all downstream devices to be either PU 2.0, PU 2.1, or a mixture. The router will be the master. You should specify *prim-xid-poll* only if all downstream devices use PU 2.1 and you want the router to be the master device of the group.

In this example, there is only one SDLC device connected to the router, and its SDLC address is specified as 0x20 in hex. This is specified by the *sdlc address* command. Recipe 15.5 shows how to configure the router for several downstream SDLC devices.

The SDLC address is purely local to this SDLC connection. So, to uniquely identify a particular remote device anywhere in the network, the mainframe needs more information. Each device is identified in VTAM in a switched major node definition using two parameters called IDBLK and IDNUM. Cisco concatenates these two identifiers together to form the Exchange of Identification (XID). The example shows how to associate a particular IDBLK/IDNUM with a particular SDLC address using the following command:

```
dlsw-branch(config-if)#sdlc xid 20 017A0006
```

The IDBLK is contained in the first three digits of the XID field, `017` in this case, and the IDNUM appears in the remaining five digits, `A0006`. Typographical errors in the XID string are some of the most common SDLC problems. Make sure that this 8-digit hexadecimal number precisely matches the IDBLK/IDNUM configured for this device in VTAM.

Next, look at the commands that work with DLSw to allow this particular device to communicate with the mainframe. First, there must be a virtual Token Ring MAC address, which is defined using the *sdlc vmac* command:

```
dlsw-branch(config-if)#sdlc vmac 4000.CCCC.0000
```

Recall that the mainframe thinks that this SDLC device is actually a Token Ring device. This command defines the fake Token Ring address that the mainframe will use to communicate with it. However, there could be several devices on this port, all of which must have unique MAC addresses. To solve this problem, the last two digits of this address are replaced by the SDLC address. So the last two hex

characters in this configuration statement must be zero. In this particular case, the *sdlc vmac* command defines a MAC address of `4000.CCCC.0000` for the serial interface. The one configured downstream device has an SDLC address of 20 (in hex), so it will appear on the mainframe's ring with the MAC address `4000.CCCC.0020`.

Just as the SDLC device must appear to be a Token Ring device to the mainframe, the mainframe must appear to be an SDLC device on this serial port. So your router must insert the FEP's MAC address into the LLC packet. This is configured with the *sdlc partner* command:

```
dlsw-branch(config-if)#sdlc partner 4000.3745.AAAA 20
```

Note that the SDLC address is specifically included in this command to allow the different devices on a multidrop SDLC circuit to communicate with different FEPs.

DLSw has to know to pick up this particular SDLC address and share it with its DLSw peer routers. You set this with the *sdlc dlsw* command:

```
dlsw-branch(config-if)#sdlc address 20
dlsw-branch(config-if)#sdlc dlsw 20
```

The router will reject this command if you have not previously defined this address with the *sdlc address* command.

In this example, we have modified one of the important SDLC timing parameters with the *sdlc poll-pause-timer* command:

```
dlsw-branch(config-if)#sdlc poll-pause-timer 200
```

In SDLC, the primary device must poll the downstream devices to see if they have something to send. The example uses this command with an argument of 200, which instructs the router to wait for a period of 200 milliseconds between these polls. The default value is 100 milliseconds. The reason for using a longer interval between polls is simply to allow the downstream devices more time to respond. In some cases—particularly if the SDLC line is long or involves modems or line drivers to reach a physically remote device—the default value is not sufficient to account for the natural latency of the line. Note, however, that this polling does not cross the DLSw bridge, only the SDLC side of the network. This ability to terminate the local polling saves bandwidth and makes the network more tolerant of WAN latency.

There is one important final note on SDLC-to-Token Ring bridging. In Recipe 15.2, when bridging Ethernet to Token Ring, we had to make a restriction on the MTU in the *dlsw remote-peer* command. This is not necessary for SDLC. The SDLC frame size is normally either 265 or 521 bytes, much smaller than even the Ethernet frame size. But the difference here is that DLSw can fragment the larger Token Ring packets when sending them out of the SDLC port. However, it cannot combine several smaller SDLC frames into a larger Token Ring frame.

15.4.4 See Also

Recipe 15.2

Top

Recipe 15.5 Configuring SDLC for Multidrop Connections

15.5.1 Problem

You want to configure a serial port for an SDLC multidrop line supporting several devices.

15.5.2 Solution

SDLC supports multidrop connections. These are serial links that connect to several downstream devices in series. Each device has its own SDLC address, which must be configured in the router. The global DLSw configuration for this example is omitted here because it is identical to the previous recipe:

```
dlsw-branch#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
dlsw-branch(config)#interface Serial1
dlsw-branch(config-if)#description Connection to three remote SDLC devices
dlsw-branch(config-if)#encapsulation sdlc
dlsw-branch(config-if)#no keepalive
dlsw-branch(config-if)#nrzi-encoding
dlsw-branch(config-if)#clock rate 4800
dlsw-branch(config-if)#sdlc role primary
dlsw-branch(config-if)#sdlc vmac 4000.CCCC.0000
dlsw-branch(config-if)#sdlc poll-pause-timer 200
dlsw-branch(config-if)#sdlc address 20
dlsw-branch(config-if)#sdlc xid 20 017A0006
dlsw-branch(config-if)#sdlc partner 4000.3745.AAAA 20
dlsw-branch(config-if)#sdlc address 21
dlsw-branch(config-if)#sdlc xid 21 017A0007
dlsw-branch(config-if)#sdlc partner 4000.3745.AAAA 21
dlsw-branch(config-if)#sdlc address 22
dlsw-branch(config-if)#sdlc xid 22 017A0008
dlsw-branch(config-if)#sdlc partner 4000.3745.AAAB 22
dlsw-branch(config-if)#sdlc slow-poll 30
dlsw-branch(config-if)#sdlc dlsw 20 21 22
dlsw-branch(config-if)#end
dlsw-branch#
```

15.5.3 Discussion

The basic router configuration is nearly the same here as it is for the single device shown in Recipe

15.4, but there are a few differences. The first difference is that all three of the SDLC addresses are configured, where the previous recipe had only one. Each SDLC address appears in four places, and you must ensure that all four are configured for all SDLC devices.

There must be an *sdlc address* command for each address, and the XID for each device must be defined with an *sdlc xid* command. You must associate each SDLC address with an FEP Token Ring MAC address using the *sdlc partner* command. In this example, the *sdlc partner* command for the third SDLC address, 22, is associated with a different FEP MAC address than the other two, just to show how this can be done. In most cases you would probably want to use the same FEP for all of the devices on a port, though.

Finally, you must associate each of these SDLC addresses with a DLSw bridge:

```
dlsw-branch(config-if)#sdlc dlsw 20 21 22
```

This tells the router to share these three specific SDLC addresses with DLSw. The router will not accept this command unless there is a matching *sdlc address* command defining each of these addresses.

Besides defining the additional SDLC addresses for the multidrop operation, this example also includes the command *sdlc slow-poll* with an argument of 30. This tells the router to ask each device for its data every 30 seconds, rather than the default 10 seconds. This is useful in multidrop configurations because it is often difficult to service all of the devices within the required time interval.

15.5.4 See Also

Recipe 15.4

Top

Recipe 15.6 Using STUN

15.6.1 Problem

You want to connect two serial devices through an IP network.

15.6.2 Solution

Serial Tunnel (STUN) provides the ability to emulate an SDLC circuit through an IP network. To simply connect two SDLC or two HDLC ports on different routers together, you can use the following:

```
Stun-A#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Stun-A(config)#interface Loopback
Stun-A(config-if)#ip address 10.1.15.5 255.255.255.252
Stun-A(config-if)#exit
Stun-A(config)#stun peer-name 10.1.15.5
Stun-A(config)#stun protocol-group 1 basic
Stun-A(config)#interface Serial1
Stun-A(config-if)#encapsulation stun
Stun-A(config-if)#nrzi-encoding
Stun-A(config-if)#clock rate 19200
Stun-A(config-if)#stun group 1
Stun-A(config-if)#stun route all tcp 10.1.15.9
Stun-A(config-if)#end
Stun-A#
```

This router could then connect its serial port to a port on a second router, configured as follows:

```
Stun-B#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Stun-B(config)#interface Loopback
Stun-B(config-if)#ip address 10.1.15.9 255.255.255.252
Stun-B(config-if)#exit
Stun-B(config)#stun peer-name 10.1.15.9
Stun-B(config)#stun protocol-group 1 basic
Stun-B(config)#interface Serial1
Stun-B(config-if)#encapsulation stun
Stun-B(config-if)#nrzi-encoding
Stun-B(config-if)#clock rate 19200
Stun-B(config-if)#stun group 1
Stun-B(config-if)#stun route all tcp 10.1.15.5
Stun-B(config-if)#end
```

```
Stun-B#
```

You can also do more interesting things with STUN. For example, if you wanted to create a virtual multidrop SDLC circuit, you could do something like this. The first router connects to the controller, while the other two hold the SDLC devices:

```
Stun-A#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Stun-A(config)#interface Loopback
Stun-A(config-if)#ip address 10.1.15.5 255.255.255.252
Stun-A(config-if)#exit
Stun-A(config)#stun peer-name 10.1.15.5
Stun-A(config)#stun protocol-group 1 sdlc
Stun-A(config)#interface Serial1
Stun-A(config-if)#encapsulation stun
Stun-A(config-if)#nrzi-encoding
Stun-A(config-if)#clock rate 19200
Stun-A(config-if)#stun group 1
Stun-A(config-if)#stun sdlc role secondary
Stun-A(config-if)#sdlc address 20
Stun-A(config-if)#sdlc address 21
Stun-A(config-if)#stun route address 20 tcp 10.1.15.9 local-ack
Stun-A(config-if)#stun route address 21 tcp 10.1.15.13 local-ack
Stun-A(config-if)#end
Stun-A#
```

Configure the second router like this:

```
Stun-B#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Stun-B(config)#interface Loopback
Stun-B(config-if)#ip address 10.1.15.9 255.255.255.252
Stun-B(config-if)#exit
Stun-B(config)#stun peer-name 10.1.15.9
Stun-B(config)#stun protocol-group 1 sdlc
Stun-B(config)#interface Serial1
Stun-B(config-if)#encapsulation stun
Stun-B(config-if)#nrzi-encoding
Stun-B(config-if)#clock rate 19200
Stun-B(config-if)#stun group 1
Stun-B(config-if)#stun sdlc role primary
Stun-B(config-if)#sdlc address 20
Stun-B(config-if)#stun route address 20 tcp 10.1.15.5 local-ack
Stun-B(config-if)#end
Stun-B#
```

Set up the third peer as follows:

```
Stun-C#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Stun-C(config)#interface Loopback
```

```
Stun-C(config-if)#ip address 10.1.15.13 255.255.255.252
Stun-C(config-if)#exit
Stun-C(config)#stun peer-name 10.1.15.13
Stun-C(config)#stun protocol-group 1 sdlc
Stun-C(config)#interface Serial1
Stun-C(config-if)#encapsulation stun
Stun-C(config-if)#nrzi-encoding
Stun-C(config-if)#clock rate 19200
Stun-C(config-if)#stun group 1
Stun-C(config-if)#stun sdlc role primary
Stun-C(config-if)#sdlc address 21
Stun-C(config-if)#stun route address 21 tcp 10.1.15.5 local-ack
Stun-C(config-if)#end
Stun-C#
```

15.6.3 Discussion

In principle, you could configure DLSw to connect two SDLC ports together across an IP network using a slight variation of Recipe 15.5. But there is a simpler way to accomplish this. Cisco IOS includes two features, STUN and Block Serial Tunnel (BSTUN). STUN is useful for connecting things such as SDLC ports, even to the extent of building virtual SDLC multidrop links. BSTUN, on the other hand, is most useful when connecting ports running the IBM Bisync protocol. BSTUN is discussed in Recipe 15.7.

The first example in this recipe shows how to simply connect two serial ports together through an IP network using an emulated serial line. This type of configuration can be useful when dealing with applications that use the serial data link protocols in a nonstandard way. It can also be useful if you have to provide a serial connection between two locations that are already in your IP network.

This example first defines a single STUN protocol group as number 1 on each router. Then, in the interface configuration blocks, this number tells STUN how to interpret the data it receives on this interface. You could define several different protocol groups supporting different protocols if required. Note that the protocol group number is purely local to the router. So what appears as protocol group number 1 on the first router could be group number 5 on the second router.

The second example shows a somewhat more complicated configuration. In this case, STUN is used to emulate not a single circuit through an IP cloud, but a multidrop circuit for use with SDLC devices. In more complex situations like this it is often better to use DLSw, but sometimes the SDLC devices need to see one another directly for one reason or another.

The only tricky part to this type of configuration is understanding which routers are *primary* and which are *secondary* for SDLC. It's a little bit easier to understand if you envision the primary as the top of the network. Everything feeds into the primary. So if a router interface connects to downstream SDLC devices, as in routers Stun-B and Stun-C, the serial port is configured as primary, because it is controlling everything downstream. On Stun-A, however, the router is acting as the network for the real

controller device, so this router's serial interface is configured as secondary.

This example also includes local acknowledgement to prevent SDLC polling from crossing the IP network:

```
Stun-C(config-if)#stun route address 21 tcp 10.1.15.5 local-ack
```

This means that the router will respond to polls on behalf of devices that are on the other end of the tunnel to save bandwidth and improve performance. Allowing acknowledgement frames to cross the IP network sometimes introduces large latencies to the SDLC network because the devices must wait longer before sending the next data frame.

15.6.4 See Also

Recipe 15.7

Top

Recipe 15.7 Using BSTUN

15.7.1 Problem

You want to connect two Bisync (BSC) devices through an IP network.

15.7.2 Solution

This pair of router configurations shows how to define a tunnel connecting two serial ports that support BSC devices:

```
BSTUN-A#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
BSTUN-A(config)#interface Loopback
BSTUN-A(config-if)#ip address 10.1.16.5 255.255.255.252
BSTUN-A(config-if)#exit
BSTUN-A(config)#bstun peer-name 10.1.16.5
BSTUN-A(config)#bstun protocol-group 1 bsc
BSTUN-A(config)#interface Serial1
BSTUN-A(config-if)#encapsulation bstun
BSTUN-A(config-if)#clock rate 19200
BSTUN-A(config-if)#bstun group 1
BSTUN-A(config-if)#bsc char-set ebcdic
BSTUN-A(config-if)#bsc secondary
BSTUN-A(config-if)#bstun route all tcp 10.1.16.9
BSTUN-A(config-if)#end
BSTUN-A#
```

The configuration of the second router is similar:

```
BSTUN-B#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
BSTUN-B(config)#interface Loopback
BSTUN-B(config-if)#ip address 10.1.16.9 255.255.255.252
BSTUN-B(config-if)#exit
BSTUN-B(config)#bstun peer-name 10.1.16.9
BSTUN-B(config)#bstun protocol-group 1 bsc
BSTUN-B(config)#interface Serial1
BSTUN-B(config-if)#encapsulation bstun
BSTUN-B(config-if)#clock rate 19200
BSTUN-B(config-if)#bstun group 1
BSTUN-B(config-if)#bsc char-set ebcdic
BSTUN-B(config-if)#bsc primary
```

```
BSTUN-B(config-if)#bstun route all tcp 10.1.16.5
BSTUN-B(config-if)#end
BSTUN-B#
```

15.7.3 Discussion

The configuration here is similar to Recipe 15.6. The main differences are in the protocols supported. The *bstun protocol-group* command in this case tells the router that BSTUN group number 1 will be passing BSC protocol data. There are several other options, including for Diebold and MDI alarm systems, as well as a generic async option.

In this recipe, the *bsc char-set* command is set to IBM's EBCDIC character set, but you can also use ASCII. The choice depends on the type of traffic you are dealing with. Mainframe Bisync applications will usually use EBCDIC. At one time, Bisync was as a popular way to connect terminals and printers to a mainframe. But (thankfully) this ancient protocol inches closer to extinction with each passing year.

The only other important point to note is that the *bstun route* command can be used to route different stations attached to the same Bisync line depending on their addresses. Bisync allows many devices to be connected to the same controller, similar to an SDLC multidrop line. For example, if you wanted station C1 going to one destination and C2 to another, you could route them separately:

```
BSTUN-A#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
BSTUN-A(config)#interface Serial1
BSTUN-A(config-if)#bstun route address C1 tcp 10.1.16.9
BSTUN-A(config-if)#bstun route address C2 tcp 10.1.16.13
BSTUN-A(config-if)#end
BSTUN-A#
```

15.7.4 See Also

Recipe 15.6

Top

Recipe 15.8 Controlling DLSw Packet Fragmentation

15.8.1 Problem

You want to control packet fragmentation in DLSw to improve throughput.

15.8.2 Solution

There are two methods for controlling packet fragmentation when using DLSw. The first is to set an MTU for the bridge, as mentioned in Recipe 15.2:

```
Router-A(config)#dlsw remote-peer 0 tcp 10.1.1.5 lf 1470 lsap-output-list 200
```

This method is used primarily when connecting media with different MTU values. However, it is also common to connect two high-MTU media such as Token Rings via an intervening network that has low-MTU links. In this situation, you should take advantage of DLSw's TCP transport by using the following command:

```
Router-A(config)#ip tcp path-mtu-discovery
```

15.8.3 Discussion

These two different commands work at different levels and accomplish different goals. The first one sets the MTU of packets that pass through the bridge. However, the DLSw packets themselves need not have the same MTU. In fact, DLSw+ is able to break up a large Token Ring packet and carry it in a series of several DLSw packets, then reassemble the large packet at the other end. The first command instructs DLSw not to accept any packets for bridging if they are larger than the specified size.

The most serious performance problems happen when the DLSw packets themselves must be fragmented in the network. In general, the DLSw routers will use the largest MTU that they can. This will usually wind up being the MTU of the first link into the IP network heading towards the router at the other end of the bridge. There could be a link along the path that can't transmit a packet this large, so a router in the middle of the network will fragment the packet according to standard TCP packet fragmentation rules. The receiving DLSw router reassembles the packet before de-encapsulating the payload packet.

This tends to be relatively inefficient, and it can cause serious throughput issues in some networks. So, to avoid the problem, you can configure both DLSw peer routers to use a clever feature of TCP called Path

MTU Discovery, which is described in RFC 1191. When the TCP connection is first made, in this case by forming a DLSw peer relationship between two routers, the routers start by figuring out the largest MTU that they can pass between them without fragmentation.

They do this by setting the Don't Fragment (DF) bit in the IP header and sending the largest packet that the interface can support. If a router somewhere in the network finds that it must fragment the packet to forward it, it will drop it instead and send back an informational ICMP "Datagram Too Big" packet to report the problem. The ICMP message includes the maximum size that it could have passed along. This allows the two end points to quickly deduce the largest packet size they can use.

TCP Path MTU Discovery is not enabled by default on Cisco routers. This command will affect all TCP sessions with this router, not just DLSw. In general, it is most effective if all of the DLSw routers have this feature enabled.

15.8.4 See Also

Recipe 15.2; RFC 1191

Top

Recipe 15.9 Tagging DLSw Packets for QoS

15.9.1 Problem

You want to set the Type of Service (TOS) field in DLSw packets to ensure that they get preferential treatment in the network.

15.9.2 Solution

In many organizations, the SNA traffic that is encapsulated in DLSw is considered both mission critical and time sensitive. Lower priority traffic should not be allowed to interfere with it. The simplest way to accomplish this is to tag these high priority packets using the standard IP Precedence field:

```
Router-A#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router-A(config)#ip local policy route-map dlswroutemap
Router-A(config)#ip route-cache policy
Router-A(config)#access-list 101 permit tcp any any eq 2065
Router-A(config)#access-list 101 permit tcp any eq 2065 any
Router-A(config)#access-list 101 permit tcp any any eq 2067
Router-A(config)#access-list 101 permit tcp any eq 2067 any
Router-A(config)#route-map dlswroutemap permit 10
Router-A(config-route-map)#match ip address 101
Router-A(config-route-map)#set ip precedence flash-override
Router-A(config-route-map)#end
Router-A#
```

15.9.3 Discussion

The most important concept here is the idea that you should set the priority of a packet at the point where it enters the network. In this case, the DLSw packet is actually created by the router, so this is the perfect place to tag it. Then every other router in the network can simply react appropriately to this priority tag without having to look deeply into the packet to figure out how important it is.

This example just shows how to set the IP Precedence field; it doesn't actually affect the forwarding behavior. You have to ensure that the interfaces on routers that need to forward this packet are configured to deal with IP Precedence properly by using Weighted Fair Queueing (WFQ) or some other appropriate queueing mechanism. Techniques for doing this are shown in Chapter 11. The actual tagging of the packet is done using policy-based routing, which is described in more detail in Chapter 5.

The IP Precedence value is set for every TCP packet with a source or destination port of 2065 or 2067. These are the two ports commonly used for DLSw traffic: 2065 is used by DLSw Version 1, and Version 2 uses 2067. Cisco's DLSw+ primarily uses 2065, although Recipe 15.10 shows an exception to this convention.

Chapter 11 demonstrates some more sophisticated options for setting QoS tag values.

15.9.4 See Also

Chapter 5; Chapter 11

Top

Recipe 15.10 Supporting SNA Priorities

15.10.1 Problem

You want DLSw to preserve and support the SNA or APPN class of service definitions for forwarding packets through your IP network.

15.10.2 Solution

To configure DLSw to follow the SNA or APPN priorities defined in the traffic flow, you must configure the peer relationship to allow multiple distinct data streams:

```
Router-A#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router-A(config)#dlsw remote-peer 0 tcp 10.1.1.5 lsap-output-list 200 priority
Router-A(config)#end
Router-A#
```

You can then go further and map the individual priority streams to specific IP Precedence values. You can define new values as follows:

```
Router-A#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router-A(config)#dlsw tos map low 0 normal 1 medium 2 high 3
Router-A(config)#end
Router-A#
```

You can also use any of the default values shown in Table 15-2Table 15-2.

Table 15-4. Default mapping of SNA priority to IP Precedence and TCP ports

IP Precedence	Value	SNA priority	DLSw TCP port
Routine	0		
Priority	1		
Immediate	2	Low	1983
Flash	3	Normal	1982

IP Precedence	Value	SNA priority	DLSw TCP port
Flash Override	4	Medium	1981
Critical	5	High	2065
Internetwork Control	6		
Network Control	7		

The TOS map does not need to match at both ends of a DLSw connection. But, if you use the *priority* option on the *dlsw remote-peer* command on one router, you must also use it on the other:

15.10.3 Discussion

Recipe 15.9 showed how to configure the router to tag the IP Precedence field in all DLSw packets for preferential treatment through the network. This example allows for the creation of four separate DLSw priority levels that follow the SNA priorities. This is useful, for example, if the SNA traffic stream includes both bulk data transfers and interactive traffic.

As soon as you enable SNA prioritization in the *dlsw remote-peer* command, DLSw forms four TCP connections instead of just one. So it is critical that you enable this option on both ends if it is required, or the remote router will simply reject the DLSw peer connection. DLSw will then start using the TCP port numbers shown in Table 15-2.

Note that the highest priority SNA traffic has an IP TOS value of 5 (Critical) by default when SNA Priority is enabled. This is not a good choice in many networks. The routers need to reserve the top two Precedence values, Internetwork Control (6) and Network Control (7), for vital functions like routing protocols. Giving high-priority SNA traffic a Precedence value of 5 means that there is no room for other high-priority traffic such as voice. This is why we have included the *dlsw tos map* command in the recipe example. This command allows you to select more appropriate TOS values for the four SNA priorities.

Whether you use this method or the one in Recipe 15.9 to set up QoS for DLSw is mostly a matter of whether you need to preserve the native SNA priority scheme. Opening four TCP connections, as in this recipe, causes the router to use more memory and CPU resources. This might become an issue on heavily loaded routers, particularly when many routers use a common central DLSw peer.

15.10.4 See Also

Recipe 15.9; Chapter 11

Top

Recipe 15.11 DLSw+ Redundancy and Fault Tolerance

15.11.1 Problem

You want to improve the fault tolerance of your DLSw network.

15.11.2 Solution

There are several things you can do to improve the reliability and fault tolerance of your network. Many of these solutions have the added benefit of improving performance. The first important thing to consider is having more than one DLSw peer router connected to the mainframe's Token Ring. In this case, you will want to make sure that you balance the load between the two peers as much as possible:

```
dlsw-branch#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
dlsw-branch(config)#source-bridge ring-group 101
dlsw-branch(config)#dlsw local-peer peer-id 10.1.2.5
dlsw-branch(config)#dlsw timer explorer-wait-time 5
dlsw-branch(config)#dlsw load-balance circuit-count
dlsw-branch(config)#dlsw remote-peer 0 tcp 10.1.1.5 lsap-output-list 200
dlsw-branch(config)#dlsw remote-peer 0 tcp 10.1.1.9 lsap-output-list 200
dlsw-branch(config)#dlsw allroute-sna
dlsw-branch(config)#end
dlsw-branch#
```

15.11.3 Discussion

This example is extremely similar to the one shown in Recipe 15.1, so we've left out the interface parts of the configuration. The main difference is the presence of a second *dlsw remote-peer* command. This points to the IP address of a second router that is on the same central Token Ring as the mainframe. So the same mainframe MAC address is visible on this branch router coming from both DLSw peers.

The *dlsw load-balance circuit-count* command tells the router to balance circuits between these two peers. For example, if several PCs are connected to the branch Token Ring and they all have SNA sessions to the mainframe, this feature will ensure that half of these sessions will follow one path, and half will follow the other. If one of these peers fails, the circuits will be reestablished through the remaining path.

The routers will not tear down circuits to balance the load, but rather will look at the current number of

circuits on each peer each time a new circuit is established. They will then use this information to decide where to put the new circuit.

The other option here is the default *round-robin* circuit balancing. Round robin has the disadvantage that it takes a very long time to rebalance the load after a failure. So we recommend using the *circuit-count* option for load balancing between two or more remote DLSw peers.

15.11.4 See Also

Recipe 15.1

Top

Recipe 15.12 Viewing DLSw Status Information

15.12.1 Problem

You want to check on DLSw status on your router.

15.12.2 Solution

This command shows the status of a DLSw peer relationship:

```
Router>show dlsw peers
Peers:                state     pkts_rx   pkts_tx  type  drops ckts TCP   uptime
 TCP 10.1.1.5        CONNECT         350       350  conf      0    0   0 02:55:03
 TCP 10.1.1.9        CONNECT         124       124  conf      0    0   0 01:17:28
 Total number of connected peers: 2
 Total number of connections:     2
```

This command looks at the status of the SNA circuits carried within these peer connections:

```
Router>show dlsw circuits
Index           local addr(lsap)    remote addr(dsap)  state          uptime
1459617889      4000.5555.a820(04)  4000.3745.aaaa(04) CONNECTED         3d05h
2097152104      4000.5555.ac21(04)  4000.3745.aaaa(04) CONNECTED         2d18h
738197600       000c.2950.aa40(14)  4000.3745.aaaa(04) CONNECTED         3d06h
2214592610      4000.aaaa.3826(04)  4000.3745.aaaa(04) CONNECTED         3d05h
2785017948      4001.bbbb.6797(04)  4000.3745.aaaa(04) CONNECTED         3d06h
Total number of circuits connected: 5
```

15.12.3 Discussion

It is important to remember the difference between a peer and a circuit. A peer relationship exists between two DLSw routers. You can bring up a peer relationship between two routers and have no application information flowing between them. A circuit, on the other hand, is an SNA connection between a device connected to one of these routers and a device connected to the other.

The *show dlsw peers* command shows only information about the peer relationship. It indicates the state of each peer connection, how many packets have been sent and received along each path, how many circuits are active, and how long the peers have been up. The example shows two peer routers, as is the case in Recipe 15.11. In this example, both peers are in a connected state, and neither has any active circuits passing through it (as shown in the ckts column).

The *show dlsw circuits* command looks at the status of the circuits. In this case, there are five active circuits. The output shows how long each of the circuits has been connected, and the MAC addresses involved. It also shows the SSAP and DSAP associated with each session, which can help you determine what circuits apply to what applications.

The index number associated with each circuit is just an identifying number that you can use to manually disconnect a particular circuit. You can clear all of the circuits at once as follows:

```
Router#clear dlsw circuits
```

Or, to clear only the second circuit listed in the example output, use the following:

```
Router#clear dlsw circuits 2097152104
```

15.12.4 See Also

Recipe 15.11

Top

Recipe 15.13 Viewing SDLC Status Information

15.13.1 Problem

You want to check the status of an SDLC device on your router.

15.13.2 Solution

You can get a lot of useful SDLC information by simply looking at the interface:

```
Router>show interface serial1
Serial1 is up, line protocol is up
  Hardware is HD64570
  Description: Connection to three remote SDLC devices
  MTU 1500 bytes, BW 1544 Kbit, DLY 20000 usec,
     reliability 255/255, txload 1/255, rxload 1/255
  Encapsulation SDLC, loopback not set
    Router link station role: PRIMARY (DCE)
    Router link station metrics:
      slow-poll 30 seconds
      T1 (reply time out) 3000 milliseconds
      N1 (max frame size) 12016 bits
      N2 (retry count) 20
       poll-pause-timer 200 milliseconds
      poll-limit-value 1
      k (windowsize) 7
      modulo 8
      sdlc vmac: 4000.CCCC.00--
  sdlc addr 20 state is CONNECT
      cls_state is CLS_IN_SESSION
      VS 0, VR 0, Remote VR 0, Current retransmit count 0
      Hold queue: 0/200 IFRAMEs 5025/618
      TESTs 0/0 XIDs 0/0, DMs 0/0 FRMRs 0/0
      RNRs 15/2 SNRMs 0/0 DISC/RDs 0/0 REJs 0/0
      Poll: clear, Poll count: 0, ready for poll, chain: 22/21
sdlc addr 21 state is CONNECT
      cls_state is CLS_IN_SESSION
      VS 0, VR 0, Remote VR 0, Current retransmit count 0
      Hold queue: 0/200 IFRAMEs 127/15
      TESTs 0/0 XIDs 0/0, DMs 0/0 FRMRs 0/0
      RNRs 1/0 SNRMs 0/0 DISC/RDs 0/0 REJs 0/0
      Poll: clear, Poll count: 0, ready for poll, chain: 20/22
sdlc addr 22 state is SNRMSENT
```

```
        cls_state is CLS_CONNECT_RSP_PEND
        VS 0, VR 0, Remote VR 0, Current retransmit count 0
        Hold queue: 0/200 IFRAMEs 25/0
        TESTs 0/0 XIDs 0/0, DMs 0/0 FRMRs 0/0
        RNRs 0/0 SNRMs 0/0 DISC/RDs 0/0 REJs 0/0
        Poll: clear, Poll count: 0, ready for poll, chain: 21/20
    Last input 00:00:00, output 00:00:00, output hang never
    Last clearing of "show interface" counters 01:05:31
Input queue: 0/75/0/0 (size/max/drops/flushes); Total output drops: 0
    Queueing strategy: fifo
    Output queue :0/40 (size/max)
    5 minute input rate 6 bits/sec, 2 packets/sec
    5 minute output rate 3 bits/sec, 1 packets/sec
       157210 packets input, 315708 bytes, 0 no buffer
       Received 287021 broadcasts, 0 runts, 0 giants, 0 throttles
      0 input errors, 0 CRC, 0 frame, 0 overrun, 0 ignored, 0 abort
       156918 packets output, 307682 bytes, 0 underruns
       0 output errors, 0 collisions, 0 interface resets
       0 output buffer failures, 0 output buffers swapped out
      0 carrier transitions
      DCD=up  DSR=up  DTR=up   RTS=down  CTS=up
```

15.13.3 Discussion

This recipe shows the output of a *show interface* command for the multidrop configuration shown in Recipe 15.5. The first line reports that the interface is "up" and line protocol is also "up." In general, these values are affected by physical issues such as whether the cabling is correctly connected, clocking, and the choice of NRZ or NRZI line coding. The other thing to look at if you have trouble bringing the line up is the duplex setting of the interface. On Cisco routers, SDLC uses full duplex by default. To change this to half-duplex, use the *sdlc hdx* command:

```
Router-A#(config)#interface Serial1
Router-A#(config-if)#sdlc hdx
```

In the output of the *show interface* command, you can immediately see that there are three SDLC addresses configured, with hexadecimal addresses 20, 21, and 22. In this particular case, only the first two stations are shown in a connected state, and the third is not responding. The router is trying to contact it, so it is listed in a "SNRMSENT" state. This means that the router has sent a Set Normal Response Mode (SNRM) request to initialize the Physical Unit (PU). Table 15-3Table 15-3 shows all of the possible states for SDLC devices.

Table 15-5. SDLC device states

State	Description

State	Description
CONNECT	Circuit initialization has completed successfully for this device.
DISCONNECT	The router is not attempting to communicate with the device.
DISCSENT	The router has sent a disconnection request to the device but has not yet received a response.
SNRMSEEN	The router has received a connection request from the device. (The router must be secondary, and the device must be primary.)
SNRMSENT	The router has sent a connection request to the device but has not yet received a response (the router must be primary to send a SNRM.)
THEMBUSY	The device has sent an RNR frame.
USBUSY	The router has sent an RNR frame.
BOTHBUSY	The router and the device are both sending RNR frames.
XIDSENT	For PU2.1 devices, this means that the router has sent the XID to the device.
XIDSTOP	For PU2.1 devices, the device has sent its XID to the router.

In normal conditions, all of your devices should be in the "CONNECT" state, so this is a good thing to check when debugging an SDLC problem. If a device is connected but you suspect that there is a problem with it, there is a useful EXEC command that is effectively an SDLC version of *ping*:

```
Router#sdlc test serial 1 20
SDLC Test for address C1 completed
Frames sent=10 Frames received=10
```

This command sends short SDLC frames out the specified interface (in this case, `Serial1`) addressed to the desired destination address (in this case, 20). If the device is online and the circuit is configured correctly, then you should see the same number of frames received as sent. By default, it sends 10 frames. You can change the number and content of these frames, but this simple form is usually sufficient.

The most common problems with an SDLC connection are mismatched XID or SDLC addresses. Sometimes you will encounter physical problems caused by either a bad cable or electrical noise. And the other most common issues are caused by clock rate problems (either too fast or too slow for the attached devices), or an incorrect choice of NRZ or NRZI line coding. It is usually best to start with the physical layer and see if the interface is coming up at all. If the interface won't come up, or comes up but won't stay up, then look for physical problems such as these. If the interface comes up, but you can't communicate with the devices, look for problems with XID values and SDLC addresses.

15.13.4 See Also

Recipe 15.5

Top

Recipe 15.14 Debugging DSLw

15.14.1 Problem

You want to debug and isolate DLSw problems.

15.14.2 Solution

The first thing to do with any DLSw issue is to verify that the peers are working correctly, as in Recipe 15.12 . If the peers are not established, then test IP connectivity with ping packets. If you can ping but the peers won't come up, then verify your configuration as in Recipe 15.1. In particular, ensure that the remote peer of each router precisely matches the local peer on the other end.

If the DLSw peers are active, check the circuits, as in Recipe 15.12

For failed circuits involving SDLC devices, check the serial interface, as in Recipe 15.13 .

For Token Ring or Ethernet devices, verify that the interface is functioning properly as in Chapter 16

If the peers are active and the interfaces look good, then there are three main things that could still be wrong. There could be a loop problem within the DLSw network. There could be a MAC address problem, or a MAC or LSAP filtering issue. Or there could be a network congestion or performance problem.

There are several useful debug commands for use with DLSw. For looking at the router-to-router DLSw transport, you can use the *debug dlsw* command:

```
dlsw-branch#debug dlsw
```

You can get other useful information about SNA and LLC2 connection problems with these debug commands:

```
dlsw-branch#debug sna state
dlsw-branch#debug llc2 state
```

15.14.3 Discussion

If your routers will not establish a DLSw peer relationship, the most common problem is simple IP connectivity. Verify that you can *ping* the address in the remote-peer statement. Note also that at most one end can use the *promiscuous* keyword to avoid configuring a *remote-peer* statement. This feature allows the router to sit and quietly wait for other routers to initiate connections. Somebody has to start the conversation.

Another common reason for failing to connect is simply a configuration mismatch between the local peer and remote peer definitions. Make sure that if you have a *remote-peer* statement, then the IP address exactly matches the address in the *local-peer* statement of the other router. It is not enough just to target any address on the remote peer router: it must be exactly the same address.

If this is correct and you can *ping* but still cannot connect, then there are two remaining possibilities. One is that there are port-specific filters or routing rules that permit *ping* , but handle DLSw packets differently. This is particularly possible when using policy-based routing in an attempt to give DLSw packets a preferred path through the network. The preferred path may have serious congestion problems, for example.

The other possibility is that there is a complete incompatibility between the versions of DLSw running at the two ends. If both are Cisco routers, then this is not going to happen. But there are still some pre-RFC 1795 DLSw implementations out there, and Cisco's DLSw+ cannot negotiate a connection with noncompliant versions.

If you are trying to debug a DLSw problem, make sure that the peers are connected:

```
dlsw-branch#show dlsw peers
Peers:              state     pkts_rx   pkts_tx  type  drops ckts TCP   uptime
 TCP 10.1.1.5     CONNECT    1348082    547505  conf      0    3   0     4w6d
 TCP 10.1.1.9     CONNECT     167013    200235  conf      0    3   0     4w5d
 TCP 10.2.1.5     DISCONN          0         0  conf      0    0   -        -
 TCP 10.2.1.9     DISCONN          0         0  conf      0    0   -        -
Total number of connected peers: 2
Total number of connections:     2
```

In this example, four peers have been configured, but only two of them are connected. A simple *ping* test would reveal in this case that the two disconnected peers are currently unreachable because of a network problem.

This command output also shows that there are currently three circuits connected to each of the active peers, for a total of six active circuits. You can look at information on the individual circuits with the *show dlsw circuits* command:

```
dlsw-branch#show dlsw circuits
Index           local addr(lsap)    remote addr(dsap)  state          uptime
2164260933      4000.53aa.2440(04)  4000.3745.aaaa(04) CONNECTED        5d00h
1627390033      4000.53aa.8420(04)  4000.3745.aaaa(04) CONNECTED        2d01h
2684354644       4000.53aa.7023(04)  4000.3745.aaaa(04) CONNECTED      09:41:36
2013265928      000c.29d0.bb41(14)  4000.3745.aaab(04) CONNECTED         4w5d
654311497       4001.001b.a1f3(04)  4000.3745.aaab(04) CONNECTED        3d18h
1342177367       4000.53aa.e3a1(04)  4000.3745.aaab(04) CONNECTED      06:09:14
Total number of circuits connected: 6
```

Here you can see that these six circuits connect to two different FEP MAC addresses. All of the DSAP

values are `0x04` , and one of the circuits has an LSAP value of `0x14` . This is useful in making sure that you have appropriate SAP filters in place.

The following shows the output of a debug trace on this same router:

```
dlsw-branch#debug dlsw
DLSw reachability debugging is on at event level for all protocol traffic
DLSw peer debugging is on
DLSw local circuit debugging is on
DLSw core message debugging is on
DLSw core state debugging is on
DLSw core flow control debugging is on
DLSw core xid debugging is on
dlsw-branch#
May 15 22:18:35.320:  DLSW Received-ctlQ : CLSI Msg : TEST_STN.Ind   dlen: 40
May 15 22:18:35.320: CSM: Received CLSI Msg : TEST_STN.Ind   dlen: 40 from
TokenRing0/0
May 15 22:18:35.320: CSM:   smac c001.001b.21aa, dmac 0020.353f.ab4a, ssap 4 , dsap 0
May 15 22:18:35.320: broadcast filter failed mac check
May 15 22:18:35.320: broadcast filter failed mac check
May 15 22:18:35.320: CSM: Write to all peers not ok - PEER_NO_CONNECTIONS
May 15 22:18:35.556:  DISP Sent : CLSI Msg : TEST_STN.Req   dlen: 46
May 15 22:18:35.556:  DISP Sent : CLSI Msg : TEST_STN.Req   dlen: 46
May 15 22:18:35.556:  DISP Sent : CLSI Msg : TEST_STN.Req   dlen: 46
May 15 22:18:35.556:  DISP Sent : CLSI Msg : TEST_STN.Req   dlen: 46
May 15 22:18:35.556:  DISP Sent : CLSI Msg : TEST_STN.Req   dlen: 46
May 15 22:18:35.556:  DISP Sent : CLSI Msg : TEST_STN.Req   dlen: 46
May 15 22:18:36.240:  DLSW Received-ctlQ : CLSI Msg : TEST_STN.Ind   dlen: 40
May 15 22:18:36.244: CSM: Received CLSI Msg : TEST_STN.Ind   dlen: 40 from
TokenRing0/0
May 15 22:18:36.244: CSM:   smac c001.001b.21aa, dmac 0060.9442.7cf3, ssap 4 , dsap 0
May 15 22:18:36.244: broadcast filter failed mac check
May 15 22:18:36.244: broadcast filter failed mac check
May 15 22:18:36.244: CSM: Write to all peers not ok - PEER_NO_CONNECTIONS
```

Note that one device with a Token Ring MAC address of `c001.001b.21aa` is attempting to make a connection to two different destination MAC addresses. These connections are being rejected with the message "broadcast filter failed mac check." This message appears because of the *dlsw icanreach mac-exclusive* command that filters out all MAC addresses from crossing the network unless they are explicitly configured on the router.

Watching this same debug trace for a little while longer reveals more interesting entries:

```
May 15 22:18:46.852: DLSw: 654311497 decr s - s:58 so:0 r:89 ro:0
May 15 22:18:46.852: DLSW Received-disp : CLSI Msg : DATA.Ind   dlen: 336
May 15 22:18:46.852: DLSw: START-FSM (654311497): event:DLC-Data.Ind state:CONNECTED
May 15 22:18:46.852: DLSw: core: dlsw_action_l(   )
May 15 22:18:46: %DLSWC-3-SENDSSP: SSP OP = 10( INFO ) to peer 10.1.1.5(2065) success
May 15 22:18:46.852: DLSw: END-FSM (654311497): state:CONNECTED->CONNECTED
May 15 22:18:46: %DLSWC-3-RECVSSP: SSP OP = 10( INFO )  from peer 10.1.1.5(2065)
```

```
May 15 22:18:46.920: DLSw: 654311497 decr r - s:58 so:0 r:88 ro:0
May 15 22:18:46.920: DLSw: START-FSM (654311497): event:WAN-INFO state:CONNECTED
May 15 22:18:46.924: DLSw: core: dlsw_action_m(   )
May 15 22:18:46.924:  DISP Sent : CLSI Msg : DATA.Req   dlen: 308
May 15 22:18:46.924: DLSw: END-FSM (654311497): state:CONNECTED->CONNECTED
May 15 22:18:48.612: DLSw: Keepalive Request sent to peer 10.1.1.9(2065))
May 15 22:18:48.628: DLSw: Keepalive Response from peer 10.1.1.9(2065)
```

In this section of the trace, you can see one of the circuits (654311497) actually sends and receives a piece of information. You can see that it uses the first of the active peers, `10.1.1.5` , and that the conversation happens on TCP port 2065. A short time later, you can see the router exchange keepalive messages with the other peer to ensure that the peer relationship remains active.

The following shows what happens when a particular circuit suddenly disconnects. In this case, we have deliberately cleared one of the circuits:

```
dlsw-branch# clear dlsw circuit 2013265928
May 15 22:18:55.412: DLSw: START-FSM (2013265928): event:ADMIN-STOP state:CONNECTED
May 15 22:18:55.412: DLSw: core: dlsw_action_r(   )
May 15 22:18:55: %DLSWC-3-SENDSSP: SSP OP = 25( HLTN )  to peer 10.1.1.5(2065)
success
May 15 22:18:55.412:  DISP Sent : CLSI Msg : DISCONNECT.Req   dlen: 4
May 15 22:18:55.412: DLSw: END-FSM (2013265928): state:CONNECTED->HALT_NOACK_PEND
May 15 22:18:55.424: SNA: Connection to Focal Point SSCP lost
May 15 22:18:55.424:  DLSW Received-ctlQ : CLSI Msg : DISCONNECT.Cfm CLS_OK dlen: 8
May 15 22:18:55.424: DLSw: START-FSM (2013265928): event:DLC-Disc.Cnf state:HALT_
NOACK_PEND
May 15 22:18:55.424: DLSw: core: dlsw_action_z(   )
May 15 22:18:55.424:  DISP Sent : CLSI Msg : CLOSE_STN.Req   dlen: 4
May 15 22:18:55.424: DLSw: END-FSM (2013265928): state:HALT_NOACK_PEND->CLOSE_PEND
May 15 22:18:55.424:  DLSW Received-ctlQ : CLSI Msg : DISCONNECT.Ind   dlen: 8
May 15 22:18:55.424: DLSw: START-FSM (2013265928): event:DLC-Disc.Ind state:CLOSE_
PEND
May 15 22:18:55.424: DLSw: END-FSM (2013265928): state:CLOSE_PEND->CLOSE_PEND
May 15 22:18:55.424:  DLSW Received-ctlQ : CLSI Msg : CLOSE_STN.Cfm CLS_OK dlen: 8
May 15 22:18:55.424: DLSw: START-FSM (2013265928): event:DLC-CloseStn.Cnf state:
CLOSE_PEND
May 15 22:18:55.428: DLSw: core: dlsw_action_y(   )
May 15 22:18:55.428: DLSw: 2013265928 to dead queue
May 15 22:18:55.428: DLSw: START-TPFSM (peer 10.1.1.5(2065)): event:CORE-DELETE
CIRCUIT state:CONNECT
May 15 22:18:55.428: DLSw: dtp_action_v(   ), peer delete circuit for peer 10.1.1.
5(2065)
May 15 22:18:55.428: DLSw: END-TPFSM (peer 10.1.1.5(2065)): state:CONNECT->CONNECT
May 15 22:18:55.428: DLSw: END-FSM (2013265928): state:CLOSE_PEND->DISCONNECTED
```

As you can see, there are several stages that the connection must go through to complete the disconnection request. First it goes into a "HALT_NOACK_PEND" state, meaning that both ends have not yet acknowledged the halt order. From there it goes into a "CLOSE_PEND" state, and finally "DISCONNECTED," as the two DLSw peer routers finally agree that the connection has been

terminated.

For one final example, here is what the *debug sna state* trace looks like for a similar disconnection and reconnection:

```
dlsw-branch#debug sna state
  SNA state change debugging is on for all PUs

dlsw-branch#clear dlsw circuit 2130706520
dlsw-branch#
May 15 22:22:57.399: SNA: LS VDLCTEST: input=Disc.Ind, Connected -> PendClose
May 15 22:22:57.403: SNA: PU VDLCTEST: input=T2ResetPu, Active -> Reset
May 15 22:22:57.403: SNA: LS VDLCTEST: input=Close.Cnf, PendClose -> Reset
May 15 22:22:57.403: SNA: Connection to Focal Point SSCP lost
May 15 22:23:27.035: SNA: LS VDLCTEST: input=StartLs, Reset -> PendConOut
May 15 22:23:33.035: SNA: LS VDLCTEST: input=ReqOpn.Cnf, PendConOut -> Xid
May 15 22:23:33.303: SNA: LS VDLCTEST: input=Connect.Ind, Xid -> ConnIn
May 15 22:23:33.303: SNA: LS VDLCTEST: input=Connected.Ind, ConnIn -> Connected
May 15 22:23:33.427: SNA: PU VDLCTEST: input=Actpu, Reset -> Active
May 15 22:23:33.427: SNA: Connection to Focal Point SSCP established
```

In this trace, you can see the exchange of XID information and the final activation of the circuit takes less than a second here. The main time lag occurs as the end devices wait a few seconds after disconnection before trying to reconnect their SNA sessions.

15.14.4 See Also

Recipe 15.1 ; Recipe 15.12 ; Recipe 15.13 ; Chapter 16

Top

Chapter 16. Router Interfaces and Media

Top

Introduction

Cisco supports a huge variety of different media types. There are over 50 different types ofinterface adapters available for the 7200 series routers alone. Of course, many of these are closely related variants such as the same OC3 card with multimode or single mode fiber connectors. The sheer variety of different media types makes it impossible for us to cover them all in any detail, so this chapter will focus instead on some of the most popular interface types. We will also look at a few interface types that have particularly interesting features or are tricky to set up properly.

We also suggest looking at some of the other chapters in this book where we have covered interface-specific material. For example, there is useful information on serial interfaces in the discussion of Frame Relay in Chapter 10. Similarly, we covered a lot of ISDN information while discussing Dial Backup in Chapter 13. And there is some discussion of both SDLC serial configuration and Token Ring features in Chapter 15, which looks at DLSw. Further, the HSRP discussion in Chapter 22 includes several useful Ethernet and Token Ring features.

Whole books have been written on each of the different media types discussed in this chapter, so we clearly can't offer a very comprehensive summary here. For information about the various serial media, refer to T1: A Survival Guide (O'Reilly). Ethernet: The Definitive Guide (O'Reilly) includes a vast amount of useful and interesting information about how Ethernet works, and Designing Large-Scale LANs (O'Reilly) includes information about other LAN protocols including Token Ring and ATM, as well as information about VLAN trunking protocols.

Top

Recipe 16.1 Viewing Interface Status

16.1.1 Problem

You want to look at the status of your router's interfaces.

16.1.2 Solution

You can look at the current status of any interface using the *show interfaces* EXEC command. With no arguments, this command will show the status of all interfaces on the router:

```
Router1#show interfaces
```

You can also look at a particular interface by including its name with the command:

```
Router1#show interfaces FastEthernet0/1
```

It is also often useful to look specifically at the IP configuration of one or all of your interfaces using the *show ip interface* command:

```
Router1#show ip interface brief
Router1#show ip interface FastEthernet0/1
```

16.1.3 Discussion

There is a huge amount of information in the output of the *show interfaces* command, and the actual content varies from interface type to interface type:

```
Router1#show interfaces FastEthernet0/1
FastEthernet0/1 is up, line protocol is up
  Hardware is AmdFE, address is 0001.9670.b781 (bia 0001.9670.b781)
  Internet address is 172.22.1.3/24
MTU 1500 bytes, BW 100000 Kbit, DLY 100 usec,
    reliability 255/255, txload 1/255, rxload 1/255
Encapsulation ARPA, loopback not set
Keepalive set (10 sec)
Full-duplex, 100Mb/s, 100BaseTX/FX
ARP type: ARPA, ARP Timeout 04:00:00
Last input 00:00:04, output 00:00:00, output hang never
Last clearing of "show interface" counters never
Input queue: 0/75/0/0 (size/max/drops/flushes); Total output drops: 0
Queueing strategy: fifo
Output queue: 0/40 (size/max)
5 minute input rate 0 bits/sec, 0 packets/sec
5 minute output rate 1000 bits/sec, 1 packets/sec
```

```
     265295 packets input, 21235441 bytes
     Received 105678 broadcasts, 0 runts, 0 giants, 0 throttles
    0 input errors, 0 CRC, 0 frame, 0 overrun, 0 ignored
    0 watchdog
     0 input packets with dribble condition detected
     1337306 packets output, 125379250 bytes, 0 underruns
     0 output errors, 0 collisions, 8 interface resets
    0 babbles, 0 late collision, 0 deferred
    0 lost carrier, 0 no carrier
     0 output buffer failures, 0 output buffers swapped out
Router1#show interfaces Serial0/0
Serial0/0 is up, line protocol is up
  Hardware is PowerQUICC Serial
  MTU 1500 bytes, BW 1544 Kbit, DLY 20000 usec,
      reliability 255/255, txload 1/255, rxload 1/255
  Encapsulation FRAME-RELAY, loopback not set
  Keepalive set (10 sec)
  LMI enq sent  108260, LMI stat recvd 108252, LMI upd recvd 0, DTE LMI up
LMI enq recvd 0, LMI stat sent  0, LMI upd sent  0
LMI DLCI 0  LMI type is ANSI Annex D  frame relay DTE
 Broadcast queue 0/64, broadcasts sent/dropped 306266/2, interface broadcasts 306266
 Last input 00:00:04, output 00:00:02, output hang never
 Last clearing of "show interface" counters 1w5d
 Input queue: 0/75/0/0 (size/max/drops/flushes); Total output drops: 0
Queueing strategy: weighted fair
 Output queue: 0/1000/64/0 (size/max total/threshold/drops)
    Conversations  0/3/256 (active/max active/max total)
    Reserved Conversations 0/0 (allocated/max allocated)
    Available Bandwidth 1158 kilobits/sec
5 minute input rate 0 bits/sec, 1 packets/sec
5 minute output rate 0 bits/sec, 0 packets/sec
    934269 packets input, 83226465 bytes, 0 no buffer
    Received 0 broadcasts, 0 runts, 0 giants, 0 throttles
   1 input errors, 0 CRC, 1 frame, 0 overrun, 0 ignored, 0 abort
    879200 packets output, 60483145 bytes, 0 underruns
    0 output errors, 0 collisions, 4 interface resets
    0 output buffer failures, 0 output buffers swapped out
   16 carrier transitions
   DCD=up  DSR=up  DTR=up  RTS=up  CTS=up

Router1#
```

The first line is one of the most important:

```
FastEthernet0/1 is up, line protocol is up
Serial0/0 is up, line protocol is up
```

This tells you that the interface is operational. There are four main possibilities here. The interface and line protocol can both be up, or they can both be down, or the interface can be up, with the line protocol down. A fourth option is that the interface can be administratively down, which means that somebody has deliberately disabled it with a *shutdown* command. There are also other options, such as the

standby and spoofing states.

If the interface is up, this means that the router is receiving the correct Layer 1 physical signaling. For the line protocol to be up as well, the router must also see correct Layer 2 information. Clearly this varies for different media types. In some cases (such as virtual software interfaces), there is no physical carrier and you will never see a loopback interface in an up/down state. But, for other types of interfaces, this can be extremely useful information for debugging problems.

The next line tells you about the interface hardware:

```
  Hardware is AmdFE, address is 0001.9670.b781 (bia 0001.9670.b781)
  Hardware is PowerQUICC Serial
```

The interface in the first case is a FastEthernet interface that uses a FastEthernet ASIC made by AMD. This information tends to be useful only when there is a known hardware bug and you want to see if your router is affected. The rest of this line is extremely useful, however, because it tells you the Ethernet MAC address. Note that it lists both the address that the router is using as well as the Burned-In Address (BIA). Most of the time these will be the same, but in Recipe 16.9 we will show how you can make your router use a different Ethernet MAC address. The second interface is a serial interface, which doesn't have a MAC address, so none is listed.

Next is the IP address, if one is configured:

```
  Internet address is 172.22.1.3/24
```

In the previous serial interface example, the interface doesn't have any Layer 3 protocol addresses, so nothing is listed. Note, however, that this command will also display IPX or AppleTalk addresses if they are configured.

Then are two lines that tell you a series of useful pieces of information about the interface's configuration and utilization. Here is the FastEthernet interface:

```
  MTU 1500 bytes, BW 100000 Kbit, DLY 100 usec,
     reliability 255/255, txload 1/255, rxload 1/255
```

And here is the corresponding information for the serial interface:

```
  MTU 1500 bytes, BW 1544 Kbit, DLY 20000 usec,
     reliability 255/255, txload 1/255, rxload 1/255
```

The first field here is the Maximum Transmission Unit (MTU), which is 1500 bytes for both interfaces. This 1500-byte MTU size is typical for IP networks, although you can change it relatively easily using the *mtu* interface configuration command, as we show in Recipe 16.2. However, using a variety of different MTU values in your network can cause performance problems due to fragmentation.

Remember that this Layer 2 MTU affects all protocols that use the interface, not just IP. This is the size of the largest Layer 2 packet that the router can send through this interface. Some media can support very

large packets, but others are more tightly constrained.

The BW field shows the configured bandwidth of the interface. It is important to note that sometimes this is not the actual throughput of the interface. On serial interfaces, the router will usually show the default value here. Even the Data Communications Equipment (DCE) serial interfaces that supply the clock signal (and therefore have a good way of estimating the theoretical maximum throughput) have this default value unless you change it manually using the bandwidth interface configuration command. This parameter is used for calculating routing protocol metrics, as well as for converting raw bit transmission rates into utilization statistics. It has nothing to do with how fast the router will transmit packets.

The reliability field in the next line is no longer commonly used. It was part of the optional metric calculation for IGRP, so it is included for historical reasons. However, the other two values, *txload* and *rxload*, are very useful. These represent the traffic utilization for the outbound and inbound interface, respectively. Both of these values are expressed as fractions of 255, rather than percentages. This may seem like an odd value, but a range from 0-255 can be conveniently represented using an 8-bit variable, which is why Cisco does it this way. Each of these values represents a fraction of the total available bandwidth shown in the BW field.

Because the *txload* and *rxload* values are rates, the router has to measure them by counting the number of bits sent and received over a finite period of time. By default, this interval is five minutes. However, you can adjust the measurement period with the *load-interval* interface configuration command:

```
Router1(config-if)#load-interval 60
```

This command takes a value in seconds as an argument. The number must be a multiple of 30 seconds, with a maximum value of 600.

The next set of lines describes the Layer 2 encapsulation on the interface. For the FastEthernet interface example, there are four lines:

```
  Encapsulation ARPA, loopback not set
  Keepalive set (10 sec)
  Full-duplex, 100Mb/s, 100BaseTX/FX
  ARP type: ARPA, ARP Timeout 04:00:00
```

In this case, the interface uses 100Mbps full-duplex Ethernet on a 100BaseTX or 100BaseFX interface. At Layer 2, it uses ARPA[1] encapsulation. There are two Layer 2 encapsulation types. The older ARPA encapsulation standard for IP packets is also called Ethernet II, and is described in RFCs 894 and 895. This standard dates to a time when Ethernet was still an experimental protocol and an IEEE standard did not exist for it.

[1] ARPA stands for the Advanced Research Projects Agency of the U.S. government, which is the agency that sponsored the initial development of the TCP/IP protocol suite.

Some time later, the IEEE officially documented the Ethernet protocol and encapsulation standards. But the IEEE standards differed from the existing ARPA standard, which was already enjoying considerable popularity in IP networks. So, rather than changing, IP continues to use the old standard, while other Layer 3 protocols such as IPX offer a choice of encapsulation types.

The interface shown in this example is configured only for IP, which is why it shows ARPA encapsulation. The serial interface in the example is using the Frame Relay protocol, so the *show interface* output includes other relevant information:

```
Encapsulation FRAME-RELAY, loopback not set
Keepalive set (10 sec)
LMI enq sent  108260, LMI stat recvd 108252, LMI upd recvd 0, DTE LMI up
LMI enq recvd 0, LMI stat sent  0, LMI upd sent  0
LMI DLCI 0  LMI type is ANSI Annex D  frame relay DTE
Broadcast queue 0/64, broadcasts sent/dropped 306266/2, interface broadcasts 306266
```

In both cases, you can see that these interfaces use the default *keepalive* value of 10 seconds. This means that these interfaces will send out a small packet every 10 seconds just to see if the interface is still working properly. If the keepalive test fails, the router will declare the interface's line protocol down.

Both of these examples also include the phrase "loopback not set." If you were to apply an external loopback test to this interface so that the router gets back all the data that it transmits, this text would change to say "loopback set." You would also see that the line protocol is up but looped at the top of the *show interface* output:

```
Serial0/0 is up, line protocol is up (looped)
```

The next few lines show how long it's been since the router has sent or received a packet on this interface, and how long it has been since you last cleared the statistical counters on this interface:

```
Last input 00:00:04, output 00:00:02, output hang never
Last clearing of "show interface" counters 1w5d
```

The queue parameters are extremely important. The serial interface in the example uses Weighted Fair Queueing, while the Ethernet uses First In First Out (FIFO). Refer to Chapter 11 for more information on different queueing strategies:

```
Input queue: 0/75/0/0 (size/max/drops/flushes); Total output drops: 0
Queueing strategy: weighted fair
Output queue: 0/1000/64/0 (size/max total/threshold/drops)
   Conversations  0/3/256 (active/max active/max total)
   Reserved Conversations 0/0 (allocated/max allocated)
   Available Bandwidth 1158 kilobits/sec
```

The *show interface* output also shows the number of packets currently in each queue (size), the maximum number of packets that the queue can hold (max), and other parameters such as drop thresholds and the number of tail drops, which vary for different queueing strategies.

Next is a large and extremely useful block of performance-related information:

```
5 minute input rate 0 bits/sec, 0 packets/sec
5 minute output rate 1000 bits/sec, 1 packets/sec
   265295 packets input, 21235441 bytes
   Received 105678 broadcasts, 0 runts, 0 giants, 0 throttles
   0 input errors, 0 CRC, 0 frame, 0 overrun, 0 ignored
   0 watchdog
   0 input packets with dribble condition detected
   1337306 packets output, 125379250 bytes, 0 underruns
   0 output errors, 0 collisions, 8 interface resets
   0 babbles, 0 late collision, 0 deferred
   0 lost carrier, 0 no carrier
   0 output buffer failures, 0 output buffers swapped out
```

The first two lines of this block show the rates for both sending and receiving data through this interface, measured in both bits per second and packets per second, and averaged over a five minute period. This is an excellent way to quickly check how much data this interface is handling. You can also get more detailed information from the exact counters that follow.

Finally, the serial interface output ends with two lines that show you information about its physical state:

```
16 carrier transitions
DCD=up  DSR=up  DTR=up  RTS=up  CTS=up
```

In this case, the interface has gone up and down 16 times since the last time the interface counters were cleared. The last line shows the current state of all of the serial signals. For other types of media, this line may contain other relevant Layer 1 information.

The DCE device raises a voltage on the Data Carrier Detect (DCD) pin to indicate that the link is ready for transmitting data. The DCE device also sends the Data Set Ready (DSR) signal when it is ready to send or receive data. When the Data Terminal Ready (DTR) signal is high, it means that the Data Terminal Equipment (DTE) device is ready to send or receive data. By default, the DTE device will wait until it sees the DSR signal, and the DCE device will wait for the DTR signal before sending any packets.

The Request To Send (RTS) signal indicates that the DTE device would like to send data and is checking to make sure that the link and the far end are ready to receive it. If everything is ready, the DCE device responds by raising the Clear To Send (CTS) signal.

So, in the above example, all of the serial signals are high, which tells you immediately that everything is working properly. Sometimes the *show interface* output will show one or more of these signals in a "down" state. This may indicate a cabling problem, or that the far-end device is simply busy right now and can't accept packets.

The *show ip interface* command shows a different set of information about the interfaces. With the

keyword *brief* , this command gives you an extremely useful listing of all of your interfaces:

```
Router1#show ip interface brief
Interface                  IP-Address      OK? Method Status  Protocol
Async65                    unassigned      YES NVRAM  down    down
FastEthernet0/0            unassigned      YES NVRAM  up      up
FastEthernet0/0.1          172.25.1.5      YES NVRAM  up      up
FastEthernet0/0.2          172.16.2.1      YES NVRAM  up      up
Serial0/0                  unassigned      YES NVRAM  up      up
Serial0/0.1                172.25.2.1      YES NVRAM  up      up
Serial0/0.2                172.20.1.1      YES manual up      up
FastEthernet0/1            172.22.1.3      YES NVRAM  up      up
Serial0/1                  10.1.1.2        YES NVRAM  up      up
Loopback0                  172.25.25.1     YES NVRAM  up      up
Router1#
```

This output shows you all of your interfaces, their IP addresses, and both the interface and protocol status. The other two columns here are labeled "OK?" and "Method." "OK?" simply refers to whether the router thinks that interface is operating correctly, while "Method" indicates how the interface acquired its IP address.

Note that the IP addresses of almost all of these interfaces were configured by NVRAM. This simply means that they have not changed since the last reboot. However, one of the interfaces, `Serial0/0.2` , was manually configured since the last reboot.

You can include a specific interface name in place of the *brief* keyword to get details on the IP configuration of this interface:

```
Router1#show ip interface FastEthernet0/1
FastEthernet0/1 is up, line protocol is up
  Internet address is 172.22.1.3/24
  Broadcast address is 255.255.255.255
  Address determined by non-volatile memory
  MTU is 1500 bytes
  Helper address is not set
  Directed broadcast forwarding is disabled
  Multicast reserved groups joined: 224.0.0.1 224.0.0.2 224.0.0.10 224.0.0.5
      224.0.0.6
Outgoing access list is not set
Inbound  access list is not set
Proxy ARP is enabled
Security level is default
Split horizon is enabled
ICMP redirects are always sent
ICMP unreachables are always sent
ICMP mask replies are never sent
IP fast switching is enabled
IP fast switching on the same interface is disabled
IP Flow switching is disabled
IP CEF switching is enabled
```

```
  IP CEF Fast switching turbo vector
  IP multicast fast switching is enabled
  IP multicast distributed fast switching is disabled
  IP route-cache flags are Fast, CEF
  Router Discovery is enabled
  IP output packet accounting is disabled
  IP access violation accounting is disabled
  TCP/IP header compression is disabled
  RTP/IP header compression is disabled
  Probe proxy name replies are disabled
  Policy routing is disabled
  Network address translation is disabled
  WCCP Redirect outbound is disabled
  WCCP Redirect inbound is disabled
  WCCP Redirect exclude is disabled
  BGP Policy Mapping is disabled
Router1#
```

In addition to these commands, there are two hidden commands that we find very useful. The first simply adds the keyword *stats* to the *show interfaces* command:

```
Router1#show interfaces FastEthernet0/1 stats
FastEthernet0/1
          Switching path    Pkts In   Chars In   Pkts Out  Chars Out
               Processor     294567   18704930     239526   22219870
            Route cache       7758     681257      48303    6129834
                  Total     302325   19386187     287829   28349704
Router1#
```

This output displays packet switching statistics for this interface. The "Processor" line shows both how many packets and how many characters the router has switched using process switching, and the "Route cache" line shows the values for fast switching. The "Pkts In" and "Chars In" columns show the values for incoming packets, while the other two columns show values for packets transmitted out through this interface.

You can get a more detailed breakdown of this switching information by adding the *switching* keyword to the show interfaces command:

```
Router1#show interfaces FastEthernet0/1 switching
FastEthernet0/1
          Throttle count          0
        Drops         RP          0         SP          0
  SPD Flushes       Fast          0        SSE          0
  SPD Aggress       Fast          0
 SPD Priority     Inputs      40510      Drops          0

     Protocol       Path    Pkts In   Chars In   Pkts Out  Chars Out
        Other    Process      11562    1022965      18730    1123800
            Cache misses          0
                     Fast          0          0          0          0
```

```
                Auton/SSE           0           0           0           0
           IP   Process         102271     8491851      220342    21066444
            Cache misses           0
                    Fast        7758      681257       48304     6129962
                Auton/SSE           0           0           0           0
          ARP   Process           1819      109140         467       31756
            Cache misses           0
                    Fast           0           0           0           0
                Auton/SSE           0           0           0           0
Router1#
```

For more information on fast switching and process switching, see Chapter 11 .

16.1.4 See Also

Recipe 16.9 ; Recipe 16.2 ; Chapter 11

Top

Recipe 16.2 Configuring Serial Interfaces

16.2.1 Problem

You want to configure a serial interface for a WAN connection.

16.2.2 Solution

When you configure a router's serial interface, you need to specify the encapsulation, the IP address, and whether the interface will be the DCE or DTE:

```
Router3#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router3(config)#interface Serial1
Router3(config-if)#description WAN Connection to Chicago
Router3(config-if)#ip address 192.168.99.5 255.255.255.252
Router3(config-if)#encapsulation hdlc
Router3(config-if)#clock rate 56000
Router3(config-if)#no shutdown
Router3(config-if)#exit
Router3(config)#exit
```

16.2.3 Discussion

There are a couple of extremely important commands in this sample configuration. The first sets the serial encapsulation protocol. In this case, we have used the High-Level Data Link Control (HDLC) protocol, which is a standard Layer 2 protocol for synchronous serial connections:

```
Router3(config-if)#encapsulation hdlc
```

In fact, HDLC is the default encapsulation type for synchronous serial interfaces on Cisco routers, so we could have omitted this command. Because it is the default, this command does not show up in the configuration:

```
Router3#show running-config interface Serial1
Building configuration...

Current configuration : 123 bytes
!
interface Serial1
 description WAN Connection to Chicago
```

```
 ip address 192.168.99.5 255.255.255.252
 clock rate 56000
end

Router3#
```

If you prefer to use a different encapsulation type, there are several other options that are appropriate in different situations. They are all set using the interface configuration command encapsulation, followed by one of the keywords shown in Table 16-1.

Table 16-1. Synchronous serial encapsulation types

Command	Description
Router3(config-if)#**encapsulation hdlc**	The default synchronous serial encapsulation type for Cisco routers.
Router3(config-if)#**encapsulation sdlc**	Sets SDLC encapsulation for use with SNA related protocols. Please refer to Chapter 15 for more information.
Router3(config-if)#**encapsulation frame-relay**	Sets Frame Relay encapsulation. Please refer to Chapter 10 for more information.
Router3(config-if)#**encapsulation ppp**	Sets the standard PPP encapsulation.
Router3(config-if)#**encapsulation x25**	Sets X.25 encapsulation.
Router3(config-if)#**encapsulation lapb**	Sets LAPB encapsulation, which is commonly used with X.25 circuits.
Router3(config-if)#**encapsulation atm-dxi**	Sets ATM encapsulation for a serial interface.
Router3(config-if)#**encapsulation smds**	Sets SMDS encapsulation.

The last two of these options allow you to do things that are somewhat unusual. For example, the *atm-dxi* keyword sets up the serial interface to use an HDLC-like protocol called Data Exchange Interface (DXI) to carry ATM cells. This requires a special external ATM Data Service Unit (ADSU) that you connect to the router via a standard serial cable.

The last option in Table 16-1 uses the *smds* keyword to enable the Switched Multi-megabit Data Service (SMDS) protocol. This is a high-speed WAN protocol used by some WAN service providers. If you use SMDS, you will require SMDS address information from your WAN vendor. SMDS is a

relatively rare protocol, which makes discussion of it beyond the scope of this book.

The second extremely important part of the example configuration is the *clock rate* command:

```
Router3(config-if)#clock rate 56000
```

This command sets the interface's line speed to 56Kbps. Most Cisco synchronous serial interfaces can support line speeds between 1.2Kbps and 4Mbps, although they generally only accept specific round number values rather than arbitrary values: 1200, 2400, 4800, 9600, 14400, 19200, 28000, 32000, 38400, 56000, 57600, 64000, 72000, 115200, 125000, 128000, 148000, 192000, 250000, 256000, 384000, 500000, 512000, 768000, 800000, 1000000, 1300000, 2000000, 4000000 and, on interfaces that support higher speeds, 8000000. Note that there is no conventional T1 setting at 1544Kbps; the closest substitutes for a T1 are 1300 or 2000Kbps. This is usually not an issue, however, because in real carrier-provided T1 circuits, the carrier's equipment generally supplies the clock.

But this *clock rate* command has another important effect. Because only DCE interfaces can supply the clock signal on synchronous serial connections, configuring this command on an interface implies that this interface is the DCE device. If you want the router to be a DTE device, you must omit the *clock rate* command.

When using serial cables, remember that DTE has a male connector, while DCE is female on the equipment end. On the router end, all Cisco high-density serial connectors look alike, regardless of whether the equipment end uses a V.35, RS 232, or any other standard serial connector. Even though the router end of these cables uses a trapezoidal shaped connector to help ensure that the right pins go into the right holes, it is remarkably easy to inadvertently connect these cables to your router upside down. If you can't get the interface to come up even though you're sure it is configured properly, try removing the cable and rotating the connector 180 degrees. You can also use the *show controller* command to see what kind of cable the router thinks you have connected:

```
Router3#show controller serial 1
HD unit 0, idb = 0x1BC830, driver structure at 0x1C1CC0
buffer size 1524  HD unit 0, V.35 DCE cable, clockrate 1300000

cpb = 0x2, eda = 0x40B4, cda = 0x40C8
RX ring with 16 entries at 0x4024000
00 bd_ptr=0x4000 pak=0x1C3D38 ds=0x402AEE8 status=80 pak_size=92
01 bd_ptr=0x4014 pak=0x1C4D58 ds=0x402E4C8 status=80 pak_size=14
02 bd_ptr=0x4028 pak=0x1C4344 ds=0x402C31C status=80 pak_size=80
<lines deleted for brevity>
Router3#
```

Unfortunately, the exact syntax and the output structure of this command varies somewhat from router to router depending on the specific serial hardware. Note that you generally have to put a space between the interface type and the interface number, as we did. If the output says something like "No DCE cable," it is likely that you have not connected the cable correctly.

If you configure a serial interface with a *clock rate* command, then connect a DTE cable, the router will ignore the *clock rate* command. And, if you already have a DTE cable connected to the interface, the router will reject this command. Furthermore, just using a null modem connector will not fool the router into thinking a DTE cable is DCE—you must use the right cable.

This particular feature confuses many people who are used to equipment that supports nonstandard DTE clocking. Cisco routers strictly enforce the relationship between cable type and clocking.

There are a few other useful commands in the configuration example:

```
Router3(config-if)#description WAN Connection to Chicago
```

The *description* command accepts an arbitrary text string that you can use for any purpose to help you manage your network. The comment in the example says where this particular circuit connects to, but you might also put other useful information such as circuit numbers or emergency contact information.

Next is the IP address. This command takes as an argument the IP address of this particular interface, and the corresponding netmask:

```
Router3(config-if)#ip address 192.168.99.5 255.255.255.252
```

In this case, because we are dealing with a simple point-to-point link that can only have two devices on it, we have used a netmask of `255.255.255.252`, or `/30` in slash notation. There is nothing to prevent you from using a larger subnet on such a link, but we recommend using a more conservative approach to help ensure that you won't run out of addresses.

Finally, we have included a *no shutdown* command:

```
Router3(config-if)#no shutdown
```

This is necessary because, by default, the router will keep all of its interfaces in an administratively disabled state until you configure them.

You can see the current state of this interface with the *show interfaces* command:

```
Router3#show interfaces Serial1
Serial1 is down, line protocol is down
  Hardware is HD64570
  Description: WAN Connection to Chicago
  Internet address is 192.168.99.5/30
  MTU 1500 bytes, BW 1544 Kbit, DLY 20000 usec,
     reliability 255/255, txload 1/255, rxload 1/255
  Encapsulation HDLC, loopback not set
  Keepalive set (10 sec)
  Last input never, output never, output hang never
  Last clearing of "show interface" counters never
  Input queue: 0/75/0/0 (size/max/drops/flushes); Total output drops: 0
  Queueing strategy: weighted fair
```

```
  Output queue: 0/1000/64/0 (size/max total/threshold/drops)
     Conversations  0/0/256 (active/max active/max total)
     Reserved Conversations 0/0 (allocated/max allocated)
     Available Bandwidth 1158 kilobits/sec
  5 minute input rate 0 bits/sec, 0 packets/sec
  5 minute output rate 0 bits/sec, 0 packets/sec
     0 packets input, 0 bytes, 0 no buffer
     Received 0 broadcasts, 0 runts, 0 giants, 0 throttles
     0 input errors, 0 CRC, 0 frame, 0 overrun, 0 ignored, 0 abort
     0 packets output, 0 bytes, 0 underruns
     0 output errors, 0 collisions, 2 interface resets
     0 output buffer failures, 0 output buffers swapped out
     0 carrier transitions
     DCD=down  DSR=down  DTR=down  RTS=down  CTS=down
Router3#
```

Note that this interface is configured, but it is not yet active because the circuit is not yet connected. It is also interesting to note that the router thinks that this interface has a bandwidth of 1544Kbps, which is indicated by the BW field. This is in spite of the fact that though we have explicitly configured this interface to be the DCE and to supply a much lower clock speed. This bandwidth number affects two things: routing protocol metrics and utilization statistics. If you want to configure a more realistic value, you can do so with the *bandwidth* interface configuration command:

```
Router3(config)#interface Serial1
Router3(config-if)#bandwidth 56
```

Note that the argument of this command must be in kilobits per second, rather than the bits per second format used for the *clock rate* command.

You can change the MTU of an interface using the *mtu* interface configuration command:

```
Router3(config)#interface Serial1
Router3(config-if)#shutdown
Router3(config-if)#mtu 18000
Router3(config-if)#no shutdown
```

Because changing the MTU will reset the interface, it is usually a good idea to issue a *shutdown* command to disable the interface before making the change (as we have done here). Bear in mind that having a larger MTU on an interface does not necessarily mean that the router will be able to take advantage of this larger frame size. If this interface never originates packets, but just acts as a link to send IP packets between two routers, then the routers will not attempt to join smaller packets together to accommodate the larger MTU. The only time this is really useful is when the interface connects directly to serial equipment that generates larger frames.

Also note that the serial interface *mtu* command is subtly different from the *ip mtu* command. The command we used here sets the Layer 2 MTU, which applies to all protocols. The *ip mtu* command only affects IP packets.

16.2.4 See Also

Chapter 10; Chapter 15

Top

Recipe 16.3 Using an Internal T1 CSU/DSU

16.3.1 Problem

You want to configure an internal CSU/DSU for a WAN connection.

16.3.2 Solution

Cisco has a variety of different types of internal CSU/DSU devices that you can install in a router. In the following example, we have configured the internal CSU to support a fractional T1 circuit:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#interface Serial0/1
Router1(config-if)#ip address 192.168.99.9 255.255.255.252
Router1(config-if)#no shutdown
Router1(config-if)#service-module t1 timeslots  1-12
Router1(config-if)#exit
Router1(config)#end
Router1#
```

16.3.3 Discussion

All of the work here is done with the *service-module* command:

```
Router1(config-if)#service-module t1 timeslots  1-12
```

This example tells the internal CSU/DSU to use the first 12 time slots of the T1 circuit. The dash character (-) tells the router to use a range of values. If you have a more complicated list of time slots, you can configure the list using a list of ranges separated by commas. For example:

```
Router1(config-if)#service-module t1 timeslots  1-3,5-19,21
```

You could even configure all of the odd-numbered time slots as follows:

```
Router1(config-if)#service-module t1 timeslots  1,3,5,7,9,11,13,15,17,19,21,23
```

If you want to use all of the time slots, you can just use the keyword *all*:

```
Router1(config-if)#service-module t1 timeslots  all
```

By default, the CSU will assume that all of these time slots are 64Kbps DS0 channels. But, if the circuit

actually uses 56Kbps channels, you can configure this by adding the *speed* keyword:

```
Router1(config-if)#service-module t1 timeslots  1-12 speed 56
```

In a production environment, the WAN carrier will usually provide the clock signal for a T1 circuit. But in some cases, particularly test and lab networks, you will need the internal CSU/DSU to act as the DCE device and supply the clock signal. You can configure the module to do this with the *clock source internal* keywords:

```
Router1(config-if)#service-module t1 clock source internal
```

By default, the router will use Binary 8 Zeros Substitution (B8ZS) line coding. While B8ZS tends to be the most common way that T1 circuits are delivered, some service providers use Alternate Mark Inversion (AMI) instead. You can configure the internal CSU to use AMI line coding as follows:

```
Router1(config-if)#service-module t1 linecode ami
```

When you use AMI line coding you have to either set the speed of each channel to 56Kbps, as we did above, or use inverted data coding:

```
Router1(config-if)#service-module t1 data-coding inverted
```

The other T1 option that you may see is the use of Super Frame (SF) instead of the default Extended Super Frame (ESF). If your WAN vendor uses SF framing, you can configure this on your router with the following command:

```
Router1(config-if)#service-module t1 framing sf
```

Some network vendors require a special Facilities Data Link (FDL) configuration. There are three options: FDL can be disabled, it can be implemented according to the ANSI T1.403 standard, or it can use the AT&T standard. To enable the ANSI standard, you use the keyword *ansi*, as follows:

```
Router1(config-if)#service-module t1 fdl ansi
```

And you can turn on the AT&T option like this:

```
Router1(config-if)#service-module t1 fdl att
```

If you are using a WIC-1DSU-T1 module, FDL is disabled by default. However, on Cisco 2524 and 2525 routers, the built-in T1 CSU/DSU uses both ANSI and AT&T options simultaneously by default, and you can't disable FDL.

It is usually a good idea to enable the *remote-alarm-enable* option:

```
Router1(config-if)#service-module t1 remote-alarm-enable
```

This option allows the CSU to send remote alarms, also called yellow alarms, to the CSU on the other end of the circuit. It does this to let the other device know that it has encountered an alarm condition

such as a framing error or loss of signal on the circuit. You should only use this option with ESF framing because it conflicts with SF framing.

The following is a real-world configuration taken from a router installed in a remote area that only supported AMI line coding, SF framing, and ANSI-standard FDL:

```
Router6(config)#interface Serial0/0
Router6(config-if)#service-module t1 framing sf
Router6(config-if)#service-module t1 linecode ami
Router6(config-if)#service-module t1 timeslots all speed 56
Router6(config-if)#service-module t1 fdl ansi
```

16.3.4 See Also

T1: A Survival Guide, by Matthew Gast (O'Reilly)

Top

Recipe 16.4 Using an Internal ISDN PRI Module

16.4.1 Problem

You want to configure an internal ISDN PRI module.

16.4.2 Solution

You can configure an ISDN PRI controller module using the *controller T1* command set as follows:

```
Router8#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router8(config)#isdn switch-type primary-dms100
Router8(config)#controller T1 0
Router8(config-controlle)#framing esf
Router8(config-controlle)#clock source line primary
Router8(config-controlle)#linecode b8zs
Router8(config-controlle)#pri-group timeslots 1-24
Router8(config-controlle)#end
Router8#
```

16.4.3 Discussion

The configuration for an ISDN PRI controller module is different from the configuration for a regular internal T1 module that we discussed in Recipe 16.3. However, the basic parameters (such as defining the framing, line coding, and so forth) are similar.

In this example, we have also defined two ISDN-specific options. The first sets the ISDN switch type to that of a Nortel DMS 100 device:

```
Router8(config)#isdn switch-type primary-dms100
```

There are several other primary-rate ISDN switch types available. If your ISDN vendor uses National ISDN switches, you would replace this command with the following:

```
Router8(config)#isdn switch-type primary-ni
```

Cisco has made a concerted effort to make the *primary-ni* option effectively switch independent in newer IOS releases. There are still some problems with some of the less common switch types, but, for the most part, you should be able to use *primary-ni* with PRI circuits from most ISDN vendors.

Note that it is often necessary to reboot the router after changing the ISDN switch type to ensure that the change takes effect.

The other ISDN-specific command in this configuration example is the *pri-group* command:

```
Router8(config-controlle)#pri-group timeslots 1-24
```

In this case, we have simply defined the available ISDN B-channels to be on channels 1 through 24, inclusive.

16.4.4 See Also

Recipe 16.3; Chapter 13

Top

Recipe 16.5 Using an Internal 56Kbps CSU/DSU

16.5.1 Problem

You want to configure an internal 56Kbps CSU/DSU.

16.5.2 Solution

The configuration for an internal 56Kbps CSU/DSU is similar to that of an internal T1 CSU/DSU:

```
Router2#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router2(config)#interface Serial0/1
Router2(config-if)#ip address 192.168.99.25 255.255.255.252
Router2(config-if)#no shutdown
Router2(config-if)#service-module 56k clock rate  9.6
Router2(config-if)#exit
Router2(config)#exit
Router2#
```

16.5.3 Discussion

There are several options available for configuring internal 56Kbps CSU/DSU cards in a router, and they all use the *service-module* command, which is similar to the T1 module. This example shows how to set the CSU/DSU to a line speed of 9600bps using the *clock rate* option. This option takes as an argument the line speed in kilobits per second, with valid options of 2.4, 4.8, 9.6, 19.2, 38.4, 56, or 64. The default line speed is 56Kbps. You can also configure the module to automatically adapt to whatever the line speed might be by using the *auto* keyword:

```
Router2(config-if)#service-module 56k clock rate auto
```

This is particularly useful if the line speed changes frequently, or if you aren't sure what value the carrier uses.

Note, however, that this *clock rate* option does not imply that the internal CSU/DSU is the clock source for the circuit. By default, the CSU/DSU assumes that the clock signal comes from the network. If you want your router to supply the clock signal instead, you must configure it to do so as follows:

```
Router2(config-if)#service-module 56k clock source internal
```

Another important option for 56Kbps modules sets the network type to be either a dialup switched-56 (also called Centrex) or a leased line. Refer to Chapter 13 for a brief discussion of switched-56 circuits. To configure the CSU/DSU to support a leased line, use the keyword *dds*, which stands for Digital Data Service:

```
Router2(config-if)#service-module 56k network-type dds
```

And you can configure the module for switched-56 using the *switched* keyword:

```
Router2(config-if)#service-module 56k network-type switched
```

However, it is important to remember that some modules are designed to support only one network type. In particular, switched-56 uses two-wire signaling, while DDS uses four wires. All of Cisco's four-wire CSU/DSU modules (such as the WIC-1DSU-56K4) can support both *switched* or *dds* options, although the default is *dds*. However, the less common two-wire modules can support only the *switched* network type, and they do not allow you to change this option.

Unfortunately, not all switched networks are created equal. Just about everybody has encountered the problem in which you hear an echo of your own voice when talking on the telephone, particularly on long-distance calls. This is annoying for voice communications, but it can be devastating for digital data transmission. In the United States, AT&T's switched 56Kbps network doesn't require echo cancelling, while Sprint's does. You can specify that your carrier behaves like the AT&T network as follows:

```
Router2(config-if)#service-module 56k network-type switched
Router2(config-if)#service-module 56k switched-carrier att
```

Or, for networks like Sprint's, use the argument *sprint* instead:

```
Router2(config-if)#service-module 56k network-type switched
Router2(config-if)#service-module 56k switched-carrier sprint
```

You can also specify *other* as the argument, which behaves exactly the same as the *att* option, but presumably makes other network vendors feel better. Note that enabling the echo cancelling feature will increase call setup times noticeably.

An interesting problem can occur on 64Kbps DDS circuits. The problem is that these types of circuits often use *in-band signaling* for error conditions. This means that there is a sequence of bits that the carrier's equipment interprets as the control code telling it to take the circuit out of service. In this type of situation, it is only a matter of time before a series of data bits in a packet duplicates the pattern and causes problems.

You can use a simple algorithm called data scrambling to overcome this problem. This algorithm just jumbles up the bit patterns to ensure the control codes never appear on the line accidentally. You can enable this feature with the *data-coding scrambled* command:

```
Router2(config-if)#service-module 56k network-type dds
Router2(config-if)#service-module 56k clock rate 64
```

```
Router2(config-if)#service-module 56k data-coding scrambled
```

We included both the network type and clock rate commands with the command that does the data scrambling to remind you that this option is useful only on 64Kbps DDS circuits. Always bear in mind that this feature does jumble user data, so you must use the same option on both ends of the circuit.

16.5.4 See Also

Recipe 16.3; Chapter 13

Top

Recipe 16.6 Configuring an Async Serial Interface

16.6.1 Problem

You want to configure a sync/async interface in asynchronous mode.

16.6.2 Solution

Cisco has a class of serial modules that can support either synchronous or asynchronous communications, as required. You can use the *physical-layer async* command to change the interface from the default synchronous to asynchronous mode:

```
Router3#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router3(config)#interface Serial1/7
Router3(config-if)#physical-layer async
Router3(config-if)#encapsulation ppp
Router3(config-if)#exit
Router3(config)#line 40
Router3(config-line)#speed 115200
Router3(config-line)#end
Router3#
```

16.6.3 Discussion

As soon as you configure the *physical-layer async* command, the router will wipe out any configuration that you might previously have had on this interface for synchronous communications.

The only real trick in this configuration is that you need to apply many of the important configuration commands to a terminal line, rather than to the interface itself. In this example, the only command we have included in the line section is the *speed* command—but there could be others. We discuss the line configuration commands for connecting to asynchronous modems in Chapter 13.

The line number in this configuration is not arbitrary. In fact, after you enable the *physical-layer async* command on the serial interface, you should break out of configuration mode and use the *show line* command to see which line the router has decided to associate with this serial interface:

```
Router3#show  line
  Tty Typ     Tx/Rx    A Modem Roty AccO AccI  Uses  Noise  Overruns  Int
    0 CTY              -    -      -    -    -     0      0     0/0       -
```

```
   40 TTY  9600/9600  -    -     -    -    -     0      0     0/0     Se1/7
   65 AUX  2400/2400  F    -     -    -    -     0      0     0/0      -
*  66 VTY             -    -     -    -    -     5      0     0/0      -
   67 VTY             -    -     -    -    -     0      0     0/0      -
   68 VTY             -    -     -    -    -     0      0     0/0      -
   69 VTY             -    -     -    -    -     0      0     0/0      -
   70 VTY             -    -     -    -    -     0      0     0/0      -

Line(s) not in async mode -or- with no hardware support:
1-39,41-64

Router3#
```

Here you can see that the router has assigned our `Serial1/7` interface to line 40. It has also set the interface to the default speed of 9600 baud. We increased this speed in our example:

```
Router3(config)#line 40
Router3(config-line)#speed 115200
```

16.6.4 See Also

Recipe 16.2; Chapter 13

Top

Recipe 16.7 Configuring ATM Subinterfaces

16.7.1 Problem

You want to configure an ATM link with PVCs that connect to several other routers.

16.7.2 Solution

Our preferred way of handling ATM PVCs is to use ATM subinterfaces. We also recommend using the IOS feature that sends ATM Operations Administration and Management (OAM) cells periodically to test the VC. Cisco provides two different syntaxes for configuring ATM PVCs. Here is an example of the older method:

```
Router2#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router2(config)#interface ATM0/0
Router2(config-if)#no ip address
Router2(config-if)#exit
Router2(config)#interface ATM0/0.1 point-to-point
Router2(config-subif)#description PVC to New York
Router2(config-subif)#ip address 192.168.250.146 255.255.255.252
Router2(config-subif)#atm pvc 1 0 60 aal5snap 10000 5000 3 oam 5
Router2(config-subif)#end
Router2#
```

In IOS 11.3, Cisco introduced a new configuration method for ATM PVCs:

```
Router2#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router2(config)#interface ATM0/0
Router2(config-if)#no ip address
Router2(config-if)#exit
Router2(config)#interface ATM0/0.1 point-to-point
Router2(config-subif)#description PVC to New York
Router2(config-subif)#ip address 192.168.250.146 255.255.255.252
Router2(config-subif)#pvc 0/60
Router2(config-if-atm-vc)#vbr-nrt 10000 5000 30
Router2(config-if-atm-vc)#oam-pvc manage 5
Router2(config-if-atm-vc)#end
Router2#
```

16.7.3 Discussion

These two configuration examples do exactly the same thing. The older syntax is considerably shorter, but the new version is a little easier to read. Let's look at the old version first:

```
Router2(config-subif)#atm pvc 1 0 60 aal5snap 10000 5000 30 oam 5
```

The first numerical argument is called the Virtual Circuit Descriptor (VCD). This is simply a unique number that you can use to distinguish this ATM VC from any others on the same router. The VCD value is strictly local to the router, so you can choose any value that you like, as long as it is unique.

The next two numbers (0 and 60, in this case) are the Virtual Path Identifier (VPI) and Virtual Circuit Identifier (VCI), respectively. These are the numbers that the ATM switch uses to identify this particular ATM VC, so you have to make sure that these match the values that the switch uses.

Then we defined the encapsulation to be ATM Adaptation Layer 5 Logical Link Control/Sub-Network Access Protocol (*aal5snap*). This is a standard method for encapsulating higher-layer protocols into ATM cells for data networks. There are other options here. We could have configured the *aal5ciscoppp* option, which is a Cisco proprietary ATM data-encapsulation format that allows you to use standard PPP authentication for a VC if required. We also could have used *aal5mux*, which is similar to aal5snap, but supports only a single higher layer protocol on each VC. The last option is Network Layer Protocol Identification (*aal5nlpid*). This option is required for interoperation with an ADSU device, as discussed in Recipe 16.2.

We generally recommend using aal5snap unless you specifically need to support PPP authentication or interoperate with ADSU devices.

The next three fields in the *atm pvc* command configure traffic shaping for this VC. In this case, we have a maximum throughput of 10,000Kbps and a committed rate of 5,000Kbps. The maximum burst length is 30 ATM cells. These parameters are optional, but if we had not included them, the router would try to send at the full line rate for this ATM circuit.

The last part of the command tells the router to send ATM Operations Administration and Management (OAM) cells through this VC periodically to test that it is working properly. There are actually several different types of OAM cells, but one type, called the F5 (or loopback) test cell, works like an IP *ping*. When the device that terminates this VC (such as the router on the far end) receives an F5 cell, it turns around and sends it back to the source. In this case, we have configured the router to send one of these OAM cells every five seconds. You can then use the *show atm pvc* command to ensure that you receive the same number of F5 cells that you send:

```
Router2#show atm pvc 0/60
ATM0/0.1: VCD: 1, VPI: 0, VCI: 60, etype:0x0, AAL5 - LLC/SNAP, Flags: 0x830
PeakRate: 10000, Average Rate: 5000, Burst Cells: 96, VCmode: 0xE000
OAM frequency: 5 second(s), InARP frequency: 15 minute(s)
InPkts: 1292959637, OutPkts: 3327374998, InBytes: 2196038015, OutBytes: 813592646
InPRoc: 19959239, OutPRoc: 24660, Broadcasts: 19481389
InFast: 1212924649, OutFast: 3297025318, InAS: 60075750, OutAS: 10843631
```

```
OAM F5 cells sent: 6804133, OAM cells received: 6740056
Status: ACTIVE
```

You can see that this VC has been active for a considerable length of time, and has sent and received a large number of F5 OAM cells. The numbers are nearly the same, but some F5 cells have clearly been lost. If you suspect a problem, it is often useful to issue this command repeatedly and watch the OAM counters increment.

You can also use the *show atm pvc* command without an argument to see all of the PVCs on the router:

```
Router2>show atm pvc
                                        AAL /          Peak   Avg.  Burst
Interface     VCD   VPI   VCI Type  Encapsulation  Kbps   Kbps  Cells Status
ATM0/0.1        1     0    60  PVC  AAL5-SNAP      10000  5000    30  ACTIVE
ATM0/0.42      42     1    42  PVC  AAL5-SNAP      10000  5000    30  ACTIVE
Router2>
```

The second example uses the newer syntax to configure the exact same parameters:

```
Router2(config-subif)#pvc 0/60
Router2(config-if-atm-vc)#vbr-nrt 10000 5000 30
Router2(config-if-atm-vc)#oam-pvc manage 5
```

The *pvc* command defines the VPI and VCI as before. Note that there is no VCD value in this example. If you like, you can configure a unique name for this VC by adding up to 16 characters before the VPI/VCI values:

```
Router2(config-subif)#pvc NEWYORK 0/60
```

The *vbr-nrt* command lets you configure the traffic shaping parameters. This assumes, of course, that this is a Variable Bit Rate Non-Real Time (VBR-NRT) PVC. If it had been an Unspecified Bit Rate (UBR) PVC, you would simply leave out the *vbr-nrt* line, because UBR is the default. For a Committed Bit Rate (CBR) PVC, the configuration specifies only the committed rate for the PVC:

```
Router2(config-if-atm-vc)#cbr 10000
```

But perhaps the biggest advantage to the new syntax is that Cisco routers can now mark ATM subinterfaces as being in a down state if the router sends out three OAM cells in a row without receiving a response. You can verify this using the *show interface* command:

```
Router2#show interfaces ATM0/0.1
ATM0/0.1 is down, line protocol is down
  Hardware is cyBus ATM
   Internet address is 172.25.25.5/30
  MTU 4470 bytes, BW 2000 Kbit, DLY 100 usec, rely 255/255, load 9/255
  Encapsulation ATM
```

Without this feature, you could power off one router and the other would still think that the PVC was still active. You can also go one step further than this and configure the router to send an SNMP trap

when it sees a PVC failure:

```
Router2(config)#snmp-server enable traps atm pvc extension oam failure loopback
```

This new feature was introduced in IOS Version 12.2(4)T.

16.7.4 See Also

Recipe 16.2

Top

Recipe 16.8 Setting Payload Scrambling on an ATM Circuit

16.8.1 Problem

You want to enable payload scrambling on your ATM circuit to prevent user data from being interpreted as an in-band control sequence.

16.8.2 Solution

The command to enable scrambling varies depending on the type of circuit. For a T3 ATM circuit, you must use the command *atm ds3-scramble*:

```
Router2#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router2(config)#interface ATM0/0
Router2(config-if)#atm ds3-scramble
Router2(config-if)#end
Router2#
```

For E3 circuits, scrambling is actually enabled by default, but you can disable it with the *no* form of the command *atm e3-scramble*:

```
Router3#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router3(config)#interface ATM0/0
Router3(config-if)#no atm e3-scramble
Router3(config-if)#end
Router3#
```

Both of the preceding commands will scramble the ATM cells at the Physical Layer Interface Module (PLIM) on the ATM interface. You can also opt to scramble only the ATM cell's payload as follows:

```
Router4#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router4(config)#interface ATM0/0
Router4(config-if)#atm scrambling cell-payload
Router4(config-if)#end
Router4#
```

16.8.3 Discussion

In Recipe 16.5, we mentioned that some types of network devices use in-band signaling to indicate that there are errors on the circuit. This can cause network problems if a switch responds to a sequence of bits in a legitimate packet. In ATM networks, however, the problem is slightly different. Here the issue is not in-band signaling, but clocking.

Some ATM equipment relies on the cell contents to maintain its clocking. If it sees a long sequence of ones or zeros, it runs the risk of losing track of where the bit boundaries are. You can avoid these types of problems by enabling scrambling on ATM cells.

However, there are two different ways to scramble an ATM cell. On physical circuits (such as DS3 and E3 circuits that support ATM services), you will often need to enable bit scrambling across the whole cell, including the ATM header. But, with other equipment (such as OC3 circuits), it is sufficient to scramble only the payload. Most Cisco ATM interfaces support payload scrambling.

If you intend to use payload scrambling, you must enable it on all of your ATM devices. For physical layer scrambling on DS3 or E3 circuits, you should consult your ATM vendor to find out if scrambling is required.

16.8.4 See Also

Recipe 16.5

Top

Recipe 16.9 Configuring Ethernet Interface Features

16.9.1 Problem

You want to force a particular Ethernet speed or duplex setting.

16.9.2 Solution

Cisco routers allow you to adjust several different Layer 1 and 2 parameters on Ethernet interfaces, depending on your specific hardware. On interfaces that support more than one medium, you can specify which media type you want to use with the *media-type* command:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#interface FastEthernet0/0
Router1(config-if)#media-type 100BaseX
Router1(config-if)#duplex full
Router1(config-if)#speed 100
Router1(config-if)#end
Router1#
```

You can also adjust parameters such as the ARP timeout interval, the MAC address, and the keepalive timer on the interface:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#interface FastEthernet0/0
Router1(config-if)#mac-address 0AAA.ABCD.0101
Router1(config-if)#arp timeout 60
Router1(config-if)#keepalive 5
Router1(config-if)#end
Router1#
```

16.9.3 Discussion

Most of the time, you will just want to use the default options when setting up Ethernet interfaces. By default, most Ethernet modules will automatically sense which media type is in use, as well as the duplex and speed settings. In this example, we have explicitly forced the router to use its 100BaseX medium, full-duplex, and 100Mbps speed settings:

```
Router1(config)#interface FastEthernet0/0
```

```
Router1(config-if)#media-type 100BaseX
Router1(config-if)#duplex full
Router1(config-if)#speed 100
```

The options for the *media-type* command depend on the module. FastEthernet modules often include a Media Independent Interface (MII), for example. By default, the router will usually select the appropriate media type by sensing the Ethernet carrier signal when you connect it to another device. But sometimes there are problems with this auto sensing, and you want to force the router to use different options. Alternatively, you may want to ensure that nobody can come along and confuse the router by plugging the other unused media interface into another device.

By default, the router will also attempt to detect both duplex and speed on FastEthernet modules. These options are not available on most conventional 10Mbps Ethernet modules, although Cisco does make full-duplex-capable 10Mbps modules for some router models. The options for the duplex command are full, half, and auto. Interestingly, you can also set the duplex options with these alternative commands:

```
Router1(config-if)#full-duplex
Router1(config-if)#half-duplex
```

There are often problems with auto negotiation. When the duplex auto negotiation process fails to work properly, the interface will still work, but you will see poor performance with a large number of errors. Since the router is generally the most important device on a LAN segment, we strongly recommend manually setting both the speed and duplex on routers to ensure that there is no possibility for confusion between the router and the LAN switch or hub.

The parameters that we modified in the second example in this recipe rarely need to be changed in practice:

```
Router1(config)#interface FastEthernet0/0
Router1(config-if)#mac-address 0AAA.ABCD.0101
Router1(config-if)#arp timeout 60
Router1(config-if)#keepalive 5
```

The default MAC address used is the BIA that we discussed in Recipe 16.1. There are a few reasons why you might want to change the MAC address. For example, sometimes a piece of legacy equipment expects to see a particular MAC address, or there might be an access list that filters traffic based on this address. In this case, it can be awkward to deal with hardware problems that force you to replace or upgrade either a router or a module. You can use the *mac-address* command to give this router the same address as the previous router.

Always be careful when changing MAC addresses. If two devices on the same network wind up with the same MAC address, it can disrupt traffic for both devices as the switches try to figure out which one is which. And, if one of these devices is a router, it can disrupt all off-segment traffic. MAC addresses must be unique.

The router uses its ARP cache table to map IP addresses to MAC addresses for devices on the local LAN

segment. By default, if the router has not seen any traffic from a particular MAC address within 14,400 seconds (4 hours), it will flush the entry from its ARP cache.

Sometimes this is not appropriate, however. For example, in many bridged environments, the bridges will remove MAC table entries after a shorter period (such as 10 minutes). In this case, if a station has been idle for more than 10 minutes, but less than 4 hours, the router will send the packet. However, the bridge must flood all segments with the traffic to find the destination device. It may be more efficient if the router simply sends out a fresh ARP query for the destination. You will probably want to reduce your ARP timeout if this is the case.

You may also want to change the ARP timeout in environments where the MAC address associated with a particular IP address changes frequently. This could happen because of DHCP, for example. Or, you will get a similar effect if there are so many ARP entries that the ARP cache frequently fills up and the router has to drop entries. In these cases, you might want to reduce the ARP timeout period using the *arp timeout* command:

```
Router1(config-if)#arp timeout 60
```

The argument to this command is a time in seconds.

Finally, the *keepalive* command allows you to control how often the router sends out a keepalive packet on the interface. This allows the router to test whether the interface is still active. By default, the router will send a keepalive packet every 10 seconds, and considers the interface to be down if it fails to see three packets in a row. If you need the router to respond more quickly to failures, you can reduce this interval using the *keepalive* command:

```
Router1(config-if)#keepalive 5
```

This command takes a single argument, which is the time interval in seconds. Giving this command an argument of 0 is the same as disabling keepalives on the interface. If you do this, the router will simply stop sending keepalive packets and assume that the interface is available no matter what happens.

Top

Recipe 16.10 Configuring Token Ring Interface Features

16.10.1 Problem

You want to configure a Token Ring interface.

16.10.2 Solution

The main thing that you need to set properly for Token Ring interfaces is the ring speed:

```
Router2#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router2(config)#interface TokenRing0
Router2(config-if)#ring-speed 4
Router2(config-if)#end
Router2#
```

You can also set the MAC address on a Token Ring interface:

```
Router2#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router2(config)#interface TokenRing0
Router2(config-if)#mac-address 0006.1111.aaaa
Router2(config-if)#end
Router2#
```

And, some routers can optionally support full-duplex Token Ring:

```
Router8#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router8(config)#interface TokenRing0/8
Router8(config-if)#full-duplex
Router8(config-if)#end
Router8#
```

16.10.3 Discussion

You can set the Token Ring's ring speed with the *ring-speed* command:

```
Router2(config)#interface TokenRing0
Router2(config-if)#ring-speed 4
```

The default for Token Ring interfaces is 16Mbps. In this example, we have reduced the speed to support the older 4Mbps standard. You need to be very careful when doing this, however, because ring speed mismatches can cause serious problems on Token Ring networks. Inserting a device with the wrong ring speed not only means that the new device can't use the ring—it will also disrupt communications among all of the other devices.

Changing MAC addresses in Token Ring networks is considerably more common than it is for Ethernet. However, as we mentioned in Recipe 16.9, you need to ensure that all MAC addresses are unique:

```
Router2(config-if)#mac-address 0006.1111.aaaa
```

Full-duplex support on Token Ring interfaces is not actually part of the IEEE standard, but many vendors have started to support it. Only a few of Cisco's Token Ring modules that are specifically designed for full-duplex support offer this command. However, if you have a router with this feature and a Token Ring switch that also supports it, full-duplex can significantly improve your network performance.

You can enable full-duplex support, where available, with the full-duplex command:

```
Router8(config-if)#full-duplex
```

The default is half-duplex.

16.10.4 See Also

Recipe 16.9

Top

Recipe 16.11 Connecting VLAN Trunks With ISL

16.11.1 Problem

You want to connect an InterSwitch Link (ISL) VLAN trunk to your router.

16.11.2 Solution

The following set of commands will allow you to connect an ISL trunk to your router:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#interface FastEthernet0/0
Router1(config-if)#no ip address
Router1(config-if)#speed 100
Router1(config-if)#full-duplex
Router1(config-if)#exit
Router1(config)#interface FastEthernet0/0.1
Router1(config-subif)#encapsulation isl 1
Router1(config-subif)#ip address 172.25.1.5 255.255.255.0
Router1(config-subif)#exit
Router1(config)#interface FastEthernet0/0.2
Router1(config-subif)#encapsulation isl 2
Router1(config-subif)#ip address 172.16.2.1 255.255.255.0
Router1(config-subif)#exit
Router1(config)#interface FastEthernet0/0.3
Router1(config-subif)#encapsulation isl 574
Router1(config-subif)#ip address 10.22.1.2 255.255.255.0
Router1(config-subif)#end
Router1#
```

16.11.3 Discussion

A *trunk* is a point-to-point link containing one or more Virtual LANs (VLANs). The main purpose of a trunk is to save physical interfaces. Without VLANs, if you wanted to connect two LAN segments into a router, you would need two Ethernet interfaces. Unfortunately, this does not scale well and it is relatively common for a single switch to support many VLANs. Using trunks to bundle the VLANs together into a single link offers some clear advantages.

While trunks carry traffic for many different VLANs, they are able to keep them separate by tagging each frame with the unique identification number for the appropriate VLAN. This allows traffic from

multiple LAN segments to share the same physical link without any danger of frames leaking onto the wrong segment. When a network device receives a tagged frame from a trunk link, it removes the tag and forwards the frame to the appropriate LAN segment as a normal frame.

When you connect a router to a trunk, it can route Layer 3 packets between the various VLANs on the trunk. Because of the VLAN tagging scheme, Layer 2 frames cannot pass from one VLAN to another. So, without a router device of some kind, there is no way to interconnect the VLANs. A configuration in which a router is connected to a trunk to allow routing between the different VLANs is often called a *router on a stick* or a one-armed router because the router routes its packets back out onto the same physical interface that it received them through.

Cisco routers support two main trunking protocols, ISL and 802.1Q. ISL is a proprietary Cisco protocol, so you can only use it between Cisco devices. Conversely, 802.1Q is an IEEE open standard that is supported by most manufacturers of network hardware. Recipe 16.12 shows how to configure an 802.1Q trunk interface on a router.

Unfortunately, the 802.1Q open standard reached the market some time after the initial demand for trunking protocols. So most manufacturers of Layer 2 switching equipment developed their own proprietary standards to fill the void. Cisco developed ISL. All newer Cisco equipment now supports 802.1Q, but there are still many older Catalyst switches that cannot support the open standard. ISL is your only choice with this equipment. In any case, Cisco's ISL support is generally more mature and stable than its 802.1Q implementations. So, while we generally recommend working with open standards where possible, ISL is still clearly the more viable option in some networks.

The first step when configuring a trunk on a router is to select a physical LAN interface to connect the trunk to. In general, we don't recommend using anything slower than a FastEthernet interface for this purpose:

```
Router1(config)#interface FastEthernet0/0
Router1(config-if)#no ip address
Router1(config-if)#speed 100
Router1(config-if)#full-duplex
```

As you can see, no special configuration is necessary on the physical interface.

Next, you need to create one subinterface on this physical interface for each different VLAN. Because each VLAN represents a different Layer 3 network, you need to give each of the subinterfaces IP addresses from the corresponding IP subnets:

```
Router1(config)#interface FastEthernet0/0.1
Router1(config-subif)#encapsulation isl 1
Router1(config-subif)#ip address 172.25.1.5 255.255.255.0
```

The *encapsulation* command associates this subinterface with a particular ISL VLAN number. ISL VLAN numbers can have any value between 1 and 1000. With this subinterface configured, the router is

now able to route packets for any devices on this VLAN, exactly as if it were directly connected to the physical LAN segment.

The *show vlans* command displays information about all of the VLANs configured on the router:

```
Router1#show vlans

Virtual LAN ID:  1 (Inter Switch Link Encapsulation)

   vLAN Trunk Interface:   FastEthernet0/0.1

   Protocols Configured:   Address:              Received:        Transmitted:
           IP              172.25.1.5              203626             342261

Virtual LAN ID:  2 (Inter Switch Link Encapsulation)

   vLAN Trunk Interface:   FastEthernet0/0.2

   Protocols Configured:   Address:              Received:        Transmitted:
           IP              172.16.2.1                 0              153807

Virtual LAN ID:  574 (Inter Switch Link Encapsulation)

   vLAN Trunk Interface:   FastEthernet0/0.3

   Protocols Configured:   Address:              Received:        Transmitted:
           IP              10.22.1.2                  0                   6

Router1#
```

We have configured this router to support three different VLANs, each with its own subinterface and IP address. The subinterface number does not necessarily need to correspond to the VLAN ID, as we have assigned VLAN number 574 to subinterface`FastEthernet0/0.3`. But if you make it a general rule to always keep the subinterface number the same as the VLAN number, it will make maintenance and troubleshooting considerably simpler in a large network.

It is useful to remember that you don't need to create a distinct subinterface for every VLAN on the switch. There may be some VLANs on this switch that you don't wish to terminate on the router. In this case, the router simply ignores any frames that are tagged with VLAN numbers that it doesn't support.

You can use the *show interfaces* command to see information about the trunking configuration of a particular subinterface:

```
Router1#show interfaces FastEthernet0/0.3
FastEthernet0/0.3 is up, line protocol is up
  Hardware is AmdFE, address is 0001.9670.b780 (bia 0001.9670.b780)
  Internet address is 10.22.1.2/24
  MTU 1500 bytes, BW 100000 Kbit, DLY 100 usec,
     reliability 255/255, txload 1/255, rxload 1/255
```

```
  Encapsulation ISL Virtual LAN, Color 574.
  ARP type: ARPA, ARP Timeout 04:00:00
Router1#
```

This shows the encapsulation type (ISL) and the VLAN number (574), along with the interface's IP address information.

16.11.4 See Also

Recipe 16.12

Top

Recipe 16.12 Connecting VLAN Trunks with 802.1Q

16.12.1 Problem

You want to connect an 802.1Q VLAN trunk directly to your router.

16.12.2 Solution

To connect an 802.1Q trunk to your router, use the following set of commands:

```
Router2#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router2(config)#interface FastEthernet1/0
Router2(config-if)#no ip address
Router2(config-if)#speed 100
Router2(config-if)#full-duplex
Router2(config-if)#exit
Router2(config)#interface FastEthernet1/0.1
Router2(config-subif)#encapsulation dot1Q  1 native
Router2(config-subif)#ip address 172.25.1.47 255.255.255.0
Router2(config-subif)#exit
Router2(config)#interface FastEthernet1/0.2
Router2(config-subif)#encapsulation dot1Q  2
Router2(config-subif)#ip address 172.25.22.4 255.255.255.0
Router2(config-subif)#exit
Router2(config)#interface FastEthernet1/0.3
Router2(config-subif)#encapsulation dot1Q  548
Router2(config-subif)#ip address 172.20.1.1 255.255.255.0
Router2(config-subif)#end
Router2#
```

To support 802.1Q features, your router must have an IOS level of at least 12.0(5)T, with the IP Plus feature set.

16.12.3 Discussion

The configuration for 802.1Q trunks is almost identical to the ISL configuration we discussed in Recipe 16.11. Please refer to that recipe for more detailed discussion of trunking in general.

The most important difference between ISL and 802.1Q trunks is that 802.1Q is an IEEE open standard. If all of your switches and routers were manufactured by Cisco, you can easily use ISL without fear of

conflict. However, if you ever need to connect a trunk link to a piece of equipment from a different vendor, you may find that 802.1Q is the only option. Further, many organizations prefer to use open standard protocols as a matter of policy, even if all of their equipment happens to come from the same vendor.

One of the important but subtle differences between ISL and 802.1Q is the number of VLANs supported. ISL supports VLAN ID numbers 1 through 1000, while 802.1Q allows values from 1 through 4095. While it is unlikely that you will ever run out of VLAN numbers with either scheme, some early IOS versions (and many early switch versions) implemented 802.1Q as if it were ISL under the covers. The result is that some older devices may only support 802.1Q VLAN ID numbers between 1 and 1000. So you may find that you are not able to use any of the higher range of values. This limitation does not exist on newer versions of Cisco equipment, but we recommend being careful to avoid interoperability problems.

You configure 802.1Q by creating subinterfaces and using the encapsulation command with the *dot1Q* keyword to assign the subinterface to a particular VLAN:

```
Router2(config)#interface FastEthernet1/0.2
Router2(config-subif)#encapsulation dot1Q  2
Router2(config-subif)#ip address 172.25.22.4 255.255.255.0
```

The number after the *dot1Q* keyword is the VLAN number that you wish to associate with this subinterface.

The only tricky part of configuring 802.1Q is defining the *native* VLAN. This often causes problems for network administrators. The native VLAN is the master VLAN assigned to the interface and it must match the native VLAN configured on the switch. The native VLAN is the only VLAN whose frames do not contain an 802.1Q VLAN tag in their Layer 2 frame headers. So, if you connect two devices through an 802.1Q trunk and they don't agree on which is the native VLAN, you will effectively merge the two native VLANs together. This is almost certainly not what you want to do.

In our example, VLAN 1 is the native VLAN. We define it using the *native* keyword:

```
Router2(config)#interface FastEthernet1/0.1
Router2(config-subif)#encapsulation dot1Q 1 native
```

The default native VLAN on many switches is VLAN number 1, but you can easily configure a different native VLAN. For example, we could use the following set of commands to reconfigure VLAN number 2 as the native VLAN:

```
Router2#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router2(config)#interface FastEthernet1/0.1
Router2(config-subif)#encapsulation dot1Q  1
Router2(config-subif)#exit
Router2(config)#interface FastEthernet1/0.2
```

```
Router2(config-subif)#encapsulation dot1Q  2 native
Router2(config-subif)#end
Router2#
```

It's important to remember that there can only be one native VLAN at a time, and that whatever you configure on the router must match what is configured on the switch. It is not safe to simply assume that VLAN number 1 will always be the native VLAN.

You can use the *show vlans* command to see information about all of VLANs configured on your router:

```
Router2#show vlans

Virtual LAN ID:  1 (IEEE 802.1Q Encapsulation)

    vLAN Trunk Interface:    FastEthernet1/0.1

 This is configured as native Vlan for the following interface(s) :
FastEthernet1/0

   Protocols Configured:   Address:              Received:        Transmitted:
           IP              172.25.1.47               4974               3149

Virtual LAN ID:  2 (IEEE 802.1Q Encapsulation)

    vLAN Trunk Interface:    FastEthernet1/0.2

   Protocols Configured:   Address:              Received:        Transmitted:
           IP              172.25.22.4                548                617

Virtual LAN ID:  548 (IEEE 802.1Q Encapsulation)

    vLAN Trunk Interface:    FastEthernet1/0.3

   Protocols Configured:   Address:              Received:        Transmitted:
           IP              172.20.1.1                   0                613

Router2#
```

This command output shows the configured VLANs and identifies which VLAN is defined as native. To view a specific 802.1Q subinterface, use the*show interface* command:

```
Router2#show interface FastEthernet1/0.1
FastEthernet1/0.1 is up, line protocol is up
  Hardware is AmdFE, address is 00e0.1e84.5131 (bia 00e0.1e84.5131)
  Internet address is 172.25.1.47/24
  MTU 1500 bytes, BW 100000 Kbit, DLY 100 usec,
     reliability 255/255, txload 1/255, rxload 1/255
  Encapsulation 802.1Q Virtual LAN, Vlan ID  1.
  ARP type: ARPA, ARP Timeout 04:00:00
Router2#
```

16.12.4 See Also

Recipe 16.11

Top

Chapter 17. Simple Network Management Protocol

Top

Introduction

Since its introduction in 1988, theSimple Network Management Protocol (SNMP) has become the most popular network management protocol for TCP/IP based networks. The IETF created SNMP to allow remote management of IP based devices using a standardized set of operations. It is now widely supported by servers, printers, hubs, switches, modems, UPS systems, and (of course) Cisco routers.

The SNMP set of standards define much more than a communication protocol used for management traffic. The standards also define how management data should be accessed and stored, as well as the entire distributed framework of SNMP agents and servers. The IETF has officially recognized SNMP as a fully standard part of the IP protocol suite. The original SNMP definition is documented in RFC 1157.

In 1993, SNMP Version 2 (SNMPv2) was created to address a number of functional deficiencies that were apparent in the original protocol. The added and improved features included better error handling, larger data counters (64-bit), improved efficiency (get-bulk transfers), confirmed event notifications (informs), and most notably, security enhancements. Unfortunately, SNMPv2 did not become widely accepted because the IETF was unable to come to a consensus on the SNMP security features.

So, a revised edition of SNMPv2 was released in 1996, which included all of the proposed enhancements except for the security facility. It is discussed in RFCs 1905, 1906, and 1907. The IETF refers to this new version as SNMPv2c and it uses the same insecure security model as SNMPv1. This model relies on passwords called *community strings* that are sent over the network as clear-text. SNMPv2c never enjoyed widespread success throughout the IP community. Consequently, most organizations continue to use SNMPv1 except when they need to access the occasional large counter variable. The IETF recently announced that SNMPv3 would be the new standard, with SNMPv1, SNMPv2, and SNMPv2c being considered purely historical.

Cisco's IOS supported SNMPv2 until Version 11.2(6)F, when Cisco began supporting SNMPv2c. Cisco continues to support SNMPv2c in every IOS version beginning with 11.2(6)F. In addition, every version of IOS has supported SNMPv1 since the earliest releases.

The compromise that became SNMPv2c left the management protocol without satisfactory security features. So, in 1998, the IETF began working on SNMPv3, which is defined in RFCs 2571-2575. Essentially, SNMPv3 is a set of security enhancements to be used in conjunction with SNMPv2c. This means that SNMPv3 is not a stand-alone management protocol and does not replace SNMPv2c or SNMPv1.

SNMPv3 provides a secure method for accessing devices using authentication, message integrity, and encryption of SNMP packets throughout the network. We have included a recipe describing how to use the SNMPv3 security enhancements (seeRecipe 17.21). Table 17-1 lists the three supported versions of

SNMP and highlights their security capabilities.

Table 17-1. SNMP versions supported by Cisco

Version	Authentication	Encryption	Description
v1	Community strings	None	Trivial authentication. Packets sent in clear-text.
v2c	Community strings	None	Trivial authentication. Packets sent in clear-text.
v3(noAuthNoPriv)	Username	None	Trivial authentication. Packets sent in clear-text.
v3(authNoPriv)	SHA or MD5 encrypted pass phrase	None	Strong authentication. Packets sent in clear-text.
v3(authPriv)	SHA or MD5 encrypted pass phrase	DES	Strong authentication. Packets are encrypted.

SNMP Management Model

SNMP defines two main types of entities, *managers* and *agents*. A manager is a server that runs network management software that is responsible for a particular network. These servers are commonly referred to as Network Management Stations (NMS). There are several excellent commercial NMS platforms on the market. Throughout this book we will refer to the freely distributed NET-SNMP system as a reference NMS.

An agent is an embedded piece of software that resides on a remote device that you wish to manage. In fact, almost every IP-capable device provides some sort of built-in SNMP agent. The agent has two main functions. First, the agent must listen for incoming SNMP requests from the NMS and respond appropriately. And second, the agent must monitor internal events and create SNMP traps to alert the NMS that something has happened. This chapter will focus mainly on how to configure the router's agent.

The NMS is usually configured to poll all of the key devices in the network periodically using *SNMP Get* requests. These are UDP packets sent to the agent on the well-known SNMP port 161. The SNMP Get request prompts the remote device to respond with one or more pieces of relevant operating information.

However, because there could be hundreds or thousands of remote devices, it is often not practical to poll a particular remote device more often than once every few minutes (and in many networks you are

lucky if you can poll each device more than a few times per hour). On a schedule like this, a remote device may suffer a serious problem that goes undetected—it's possible to crash and reboot in between polls from the NMS. So, on the next poll, the NMS will see everything operating normally and never know that it completely missed a catastrophe.

Therefore, an SNMP agent also has the ability to send information using an *SNMP trap* without having to wait for a poll. A trap is an unsolicited piece of information, usually representing a problem situation (although some traps are more informational in nature). Traps are UDP packets sent from the agent to the NMS on the other well-known SNMP port number, 162. There are many different types of traps that an agent can send, depending on what type of equipment it manages. Some traps represent non-critical issues. It is often up to the network administrator to decide which types of traps will be useful.

The NMS does not acknowledge traps, and since traps are often sent to report network problems, it is not uncommon for trap reports to get lost and never make it to the NMS. In many cases, this is acceptable because the trap represents a transient transmission problem that the NMS will discover by other means if this trap is not delivered. However, critical information can sometimes be lost when a trap is not delivered.

To address this shortcoming, SNMPv2c and SNMPv3 include another type of packet called an *SNMP inform*. This is nearly identical to a standard trap, except that the SNMP agent will wait for an acknowledgement. If the agent does not receive an acknowledgement within a certain amount of time, it will attempt to retransmit the inform.

SNMP informs are not common today because SNMPv2c was never widely adopted. However, SNMPv3 also includes informs. Since SNMPv3 promises to become the mainstream SNMP protocol, it seems inevitable that enhancements such as SNMP informs will start to be more common.

MIBs and OIDs

SNMP uses a special tree structure called a Management Information Base (MIB) to organize the management data. People will often talk about different MIBs, such as the T1 MIB, or an ATM MIB. In fact, these are all just branches or extensions of the same global MIB tree structure. However, the relative independence of these different branches makes it convenient to talk about them this way.

A particular SNMP agent will care only about those few MIB branches that are relevant to the particular remote device this agent runs on. If the device doesn't have any T1 interfaces, then the agent doesn't need to know anything about the T1 branch of the global MIB tree. Similarly, the NMS for a network containing no ATM doesn't need to be able to resolve any of the variables in the ATM branches of the MIB tree.

The MIB tree structure is defined by a long sequence of numbers separated by dots, such as `.1.3.6.1.2.1.1.4.0`. This number is called an Object Identifier (OID). Since we will be working with OID strings throughout this chapter, it is worthwhile to briefly review how they work and what

they mean.

The OID is a numerical representation of the MIB tree structure. Each digit represents a node in this tree structure. The trunk of the tree is on the left; the leaves are on the right. In the example string, `.1.3.6.1.2.1.1.4.0`, the first digit, `.1`, signifies that this variable is part of the MIB that is administered by the International Standards Organization (ISO). There are other nodes at this top level of the tree. The International Telephone and Telegraph Consultative Committee (CCITT) administers the `.0` tree structure. The ISO and CCITT jointly administer `.2`.

The first node under the ISO MIB tree of this example is `.3`. The ISO has allocated this node for all other organizations. The U.S. Department of Defense (DOD) is designated by the branch number `.6`. The DOD, in turn has allocated branch number `.1` for the Internet Activities Board (IAB). So, just about every SNMP MIB variable you will ever see will begin with `.1.3.6.1`.

There are four commonly used subbranches under the IAB (also called simply "Internet") node. These are designated directory (1), mgmt (2), experimental (3) and private (4). The directory node is seldom used in practice. The mgmt node is used for all IETF-standard MIB extensions, which are documented in RFCs. This would include, for example, the T1 and ATM examples mentioned earlier. However, it would not include any vendor-specific variables such as the CPU utilization on a Cisco router. SNMP protocol and application developers use the experimental subtree to hold data that is not yet standard. This allows you to use experimental MIBs in a production network without fear of causing conflicts. Finally, the private subtree contains vendor specific MIB variables.

Before returning to the example, we want to take a brief detour down the private tree, because many of the examples in this book include Cisco-specific MIB variables. A good example of a Cisco MIB variable is `.1.3.6.1.4.1.9.2.1.8.0`, which gives the amount of free memory in a Cisco router. There is only one subtree under the private node, and it is called enterprises, `.1.3.6.1.4.1`. Of the hundreds of registered owners of private MIB trees, Cisco is number 9, so all Cisco-specific MIB extensions begin with `.1.3.6.1.4.1.9`.

Referring again to the previous example string (`.1.3.6.1.2.1.1.4.0`), you can see this represents a variable in the *mgmt* subtree, `.1.3.6.1.2`. The next digit is `.1` here, which represents an SNMP MIB variable.

The following digit, `.1`, refers to a specific group of variables, which, in the case of mgmt variables, would be defined by an RFC. In this particular case, the value `.1` refers to the *system* MIB, which is detailed in RFC 1450.

From this level down, a special naming convention is adopted to help you to remember which MIB you are looking at. The names of every variable under the *system* node begin with "sys". They are sysDescr (1), sysObjectID (2), sysUpTime (3), sysContact (4), sysName (5), sysLocation (6), sysServices (7), sysORLastChange `(8)`, and sysORTable `(9)`. You can find detailed descriptions of what all of these mean in RFC 1450.

In fact, reading through MIB descriptions is not only an excellent way to understand the hierarchical structure of the MIB, but it's also extremely useful when you are trying to decide what information you can and should be extracting from your equipment.

In the example string, `.1.3.6.1.2.1.1.4.0`, the value is `.4`, for `sysContact`. The following `.0` tells the agent to send the contents of this node, rather than treating it as the root of further subtrees. So the OID string uniquely identifies a single piece of information. In this case, that information is the contact information for the device.

Top

Recipe 17.1 Configuring SNMP

17.1.1 Problem

You want to set up basic SNMP services on a router.

17.1.2 Solution

To enable read-only SNMP services, use the following configuration command:

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#snmp-server community ORARO ro
Router(config)#end
Router#
```

To enable read-write SNMP services, use the following command:

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#snmp-server community ORARW rw
Router(config)#end
Router#
```

> It is extremely risky to enable read-write SNMP services without the appropriate security controls. Please read the following discussion section before implementing this recipe.

Starting with IOS Version 12.0(3)T, Cisco introduced a new system for configuring SNMP services using the *snmp-server group* and *snmp-server user* configuration commands. Use the following commands to enable read-only SNMP services with this new method:

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#snmp-server group COOKRO v1
Router(config)#snmp-server user TESTRO1 COOKRO  v1
Router(config)#snmp-server group BOOKRO v2c
Router(config)#snmp-server user TESTRO2 BOOKRO  v2c
Router(config)#end
```

17.1.3 Discussion

SNMP services are disabled by default on all Cisco routers. The examples highlighted in the solutions section show only how to configure the router to allow inbound SNMP services so that it will respond to SNMP Get and Set requests. These configuration examples do not enable SNMP traps or informs, which we discuss in Recipe 17.13.

When inbound SNMP services are enabled, the router starts to listen for incoming SNMP requests on UDP port 161. It is important to note that these methods enable SNMPv1 and SNMPv2c only (SNMPv3 is covered in Recipe 17.22).

The first example shows the older method of enabling SNMP services. It uses the *snmp community* command to simultaneously enable both SNMPv1 and SNMPv2c:

```
Router(config)#snmp-server community ORARO ro
```

Cisco documentation often refers to this as bilingual SNMP support because it allows the router to speak both SNMP languages (or versions).

The *show snmp group* command gives details on exactly what SNMP versions are enabled, as well as the security models they use. Running this command after implementing the first two configuration examples gives the following output:

```
Router#show snmp group
groupname: ORARO                          security model:v1
readview :v1default                       writeview: <no writeview specified>
notifyview: <no notifyview specified>
row status: active

groupname: ORARO                          security model:v2c
readview :v1default                       writeview: <no writeview specified>
notifyview: <no notifyview specified>
row status: active

groupname: ORARW                          security model:v1
readview :v1default                       writeview: v1default
notifyview: <no notifyview specified>
row status: active

groupname: ORARW                          security model:v2c
readview :v1default                       writeview: v1default
notifyview: <no notifyview specified>
row status: active
```

This shows that the groups we have configured each created two entries: one for SNMPv1 support, and the other for SNMPv2c. This is possible because SNMPv1 and SNMPv2c use the same community string authorization model. Therefore, the router is capable of responding to either version of SNMP.

When using the *snmp-server group* command for enabling SNMP services, you can create an SNMP

group entry that belongs to a single SNMP security model (SNMPv1 or SNMPv2c). Cisco added this command to support SNMPv3 services, which may eventually replace the legacy method.

Both methods for enabling SNMP services assign a community string that acts as a password of sorts to protect access. SNMP services run on a well-known UDP port, so the router needs this password to help prevent unauthorized access. The router will simply discard any SNMP requests that contain incorrect SNMP community strings. It is important to note that in both Version 1 and 2c, SNMP transmits these community strings through the network in clear-text, making it relatively insecure.

You can configure the router for either read-only or read-write SNMP service. Read-only access means that users can view the router's MIB tree. This is relatively benign, although information about the router's configuration could be useful to someone planning an attack on your network. Read-write access, on the other hand, permits users to change some of the values in the MIB tree. A user with read-write access could potentially wreak havoc by disabling IP routing, disabling interfaces, erasing router flash, downloading router configurations, uploading configuration commands, or making a variety of other dangerous changes.

Be extremely careful when providing SNMP write access. If SNMP write access is not absolutely required, we recommend disabling it. Far too many organizations automatically enable full SNMP write access without regard for the dangers of possible unauthorized changes.

If write access is required, you will first need to define SNMP views as described in Recipe 17.7. But a better solution is to simply use SNMPv3, which we discuss in Recipe 17.22. SNMPv3 offers advanced authentication and encryption services that ensure safe delivery over insecure networks. Unfortunately, SNMPv3 is still a relatively young standard, and many NMS systems don't yet support it.

So, in those cases where SNMP Version 1 or 2c write access is an absolute requirement, we recommend using the security features described in Recipe 17.5, Recipe 17.6, and Recipe 17.7. These describe how to implement SNMP ACLs, limit SNMP views, and restrict SNMP TFTP access to help to reduce the risk of attack.

17.1.4 See Also

Recipe 17.13; Recipe 17.5; Recipe 17.6; Recipe 17.7; Recipe 17.22

Top

Recipe 17.2 Extracting Router Information via SNMP Tools

17.2.1 Problem

You wish to extract or change router information via SNMP.

17.2.2 Solution

To extract router information via SNMP, we will use the suite of SNMP tools provided with the NET-SNMP toolkit (see Appendix A for more details).

Use *snmpget* to extract a single MIB object from the router's MIB tree. This example uses *snmpget* to extract the router's system contact information:

```
freebsd% snmpget -v1 -c ORARO Router .1.3.6.1.2.1.1.4.0
system.sysContact.0 = Helpdesk   800-555-2992
```

Use *snmpset* to alter MIB objects within the router's MIB tree. The next example demonstrates how to modify MIB variables, using *snmpset* to change the system contact information:

```
freebsd% snmpset -v1 -c ORARW Router .1.3.6.1.2.1.1.4.0 s " Ian Brown 555-1221 "
system.sysContact.0 = Ian Brown 555-1221
freebsd% snmpget -v1 -c ORARO Router sysContact.0
system.sysContact.0 = Ian Brown 555-1221
```

The *snmpwalk* utility extracts a series of MIB objects from the router's MIB tree. This example uses *snmpwalk* to extract all of the router's interface descriptions:

```
freebsd% snmpwalk -v1 -c ORARO Router ifDescr
interfaces.ifTable.ifEntry.ifDescr.1  =  "Ethernet0"
interfaces.ifTable.ifEntry.ifDescr.2  =  "Serial0"
interfaces.ifTable.ifEntry.ifDescr.3  =  "Serial1"
interfaces.ifTable.ifEntry.ifDescr.4  =  "Null0"
interfaces.ifTable.ifEntry.ifDescr.5  =  "Loopback0"
interfaces.ifTable.ifEntry.ifDescr.6  =  "Serial0.1"
freebsd%
```

17.2.3 Discussion

In this recipe, we use the suite of SNMP tools provided by the NET-SNMP project (formerly UCD-SNMP) to demonstrate basic SNMP functionality. NET-SNMP provides a variety of useful SNMP

tools that you can run from the command-line interface of any Unix or Windows workstation. This software is freely distributed and is available on a variety of platforms, which makes it extremely popular for scripts of all shapes and sizes. We consider NET-SNMP to be a sort of Swiss army knife of SNMP that wonderfully illustrates the usefulness of SNMP for working with Cisco routers. Of course, many commercial software vendors also provide SNMP tools that are equally effective and frequently include a graphical user interface. The underlying concepts remain the same, even if the command syntax differs. In some cases it is easier to do the types of SNMP commands shown in this recipe using a graphical user interface, rather than a command-line utility.

NET-SNMP provides a set of SNMP utilities for performing various useful SNMP functions. This recipe used three of the most basic tools.

- snmpget gets a single MIB object and displays its contents. To do this, it sends the router an SNMP "get" request for a particular MIB object. The router responds with the value of the MIB object, if present. The command syntax for SNMPv1 and SNMPv2c queries is:
```
snmpget  [options] {-c <community-string>} <hostname> [<MIB Object or OID>]
```

- snmpwalk asks the router for a group of related MIB objects and displays their contents. It does this by sending the router a series of SNMP "get-next" commands to list all available MIB objects under the specified node in an MIB tree. The router will continue to respond to the server's "get-next" requests until it reaches the end of the MIB subtree. The command syntax is as follows:
```
snmpwalk [options] {-c <community-string>} <hostname>  [<MIB Object or OID>]
```

Note that leaving out the MIB object or OID causes *snmpwalk* to walk the entire MIB tree. This can cause CPU overload problems on the router, as well as congestion problems on low-speed links.

- snmpset modifies the contents of a MIB object and displays the changed variable, if successful. It works by sending the router an SNMP "set" request for the specified MIB object. If the requested change is legal, the router will change the value of the corresponding MIB variable and respond back.

  Not all MIB entries can be changed by an SNMP set. For example, it doesn't make sense to change the physical media type of an interface. And, of course, the router has to be configured to allow SNMP read-write access. The command syntax is as follows:
```
snmpset [options] {-c <community>} <hostname> [<objectID> <type> <value>]
```

Most NMS systems have similar commands that you can access from the command line and use in scripts. See your software documentation for details.

Table 17-1Table 17-1 shows a number of useful MIB entries and their associated OID numbers. Several of these variables are Cisco-specific, and will not make sense if used on equipment from other vendors.

Table 17-2. Common Cisco router SNMP MIB entries

MIB name	Description	OID
sysName	Hostname	.1.3.6.1.2.1.1.5.0
sysUpTime	Uptime	.1.3.6.1.2.1.1.3.0
sysDescr	System Description	.1.3.6.1.2.1.1.1.0
sysContact	System Contact	.1.3.6.1.2.1.1.4.0
sysLocation	System Location	.1.3.6.1.2.1.1.6.0
ciscoImageString.5	IOS Version	.1.3.6.1.4.1.9.9.25.1.1.1.2.5
avgBusy1	1-Minute CPU Util.	.1.3.6.1.4.1.9.2.1.57.0
avgBusy5	5-Minute CPU Util.	.1.3.6.1.4.1.9.2.1.58.0
freeMem	Free memory	.1.3.6.1.4.1.9.2.1.8.0
ciscoImageString.4	IOS feature set	.1.3.6.1.4.1.9.9.25.1.1.1.2.4
whyReload	Reload reason	.1.3.6.1.4.1.9.2.1.2.0

A complete listing of Cisco-supported MIBs are located at http://www.cisco.com/public/sw-center/netmgmt/cmtk/mibs.shtml. Note that this includes a huge amount of information. However, with a little time and effort, you should be able to find a way to extract exactly the information you need.

You can extract the same MIB objects using SNMPv2c:

```
freebsd% snmpget -v 2c -c ORARO Router sysContact.0
system.sysContact.0 = Ian Brown 416-555-2943
freebsd%
```

The only difference in this example is that we specified the SNMP version number as part of the *snmpget* command syntax. This is useful because SNMPv2c introduced 64-bit counters. Cisco supports a small number of MIB objects that can only be accessed using SNMPv2c (or SNMPv3). One such MIB object is *ifHCInOctets*:

```
Freebsd% snmpwalk -v 2c -c ORARO Router ifHCInOctets
ifHCInOctets.7 = Counter64: 145362298
ifHCInOctets.8 = Counter64: 85311547
Freebsd%
```

This MIB object counts the number of inbound bytes (octets) received by an interface. The older SNMPv1 *ifInOctets* MIB object counts exactly the same thing, but uses a 32-bit variable to hold the number. The newer object does not roll over to zero as often, making it more useful for high-speed interfaces. If you attempt to get one of these 64-bit counter objects using SNMPv1, the query will fail.

17.2.4 See Also

Recipe 17.1; Recipe 17.21; Appendix A

Top

Recipe 17.3 Recording Important Router Information for SNMP Access

17.3.1 Problem

You want to record important information such as physical locations, contact names, and serial numbers for later SNMP access.

17.3.2 Solution

To record important physical information regarding the router, use the following commands:

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#snmp-server contact Ian Brown 416-555-2943
Router(config)#snmp-server location 999 Queen St. W., Toronto, Ont .
Router(config)#snmp-server chassis-id JAX123456789
Router(config)#end
Router#
```

17.3.3 Discussion

It is an extremely good network management practice to add useful information such as contact names, router locations, and serial numbers directly into the router configuration. This information can be extracted later using *snmpget* requests, either directly or invoked from scripts that make output easier to understand. This is true not only for Cisco routers—it's good practice to configure serial numbers and locations on all equipment so that they can be read with SNMP. When a field technician swaps or moves a piece of equipment, they can update this information. This makes it easy to verify the information stored in your central equipment inventory database.

When you save the contact name, router location, and serial number in the router configuration, it is also easy to extract from the command line and configuration files:

```
Router#show snmp
Chassis:  JAX123456789
Contact: Ian Brown 416-555-2943
Location: 999 Queen St. W., Toronto, Ontario
417 SNMP packets input
    0 Bad SNMP version errors
     141 Unknown community name
     3 Illegal operation for community name supplied
```

```
     0 Encoding errors
     224 Number of requested variables
     49 Number of altered variables
     224 Get-request PDUs
     0 Get-next PDUs
     52 Set-request PDUs
299 SNMP packets output
     0 Too big errors (Maximum packet size 1500)
     3 No such name errors
     0 Bad values errors
     0 General errors
     276 Response PDUs
     23 Trap PDUs

SNMP logging: enabled
     Logging to 172.25.1.1.162, 0/10, 21 sent, 2 dropped.
Router#
```

You can also extract individual pieces of information using the following commands:

```
Router#show snmp contact
Ian Brown 416-555-2943
Router#show snmp location
999 Queen St. W., Toronto, Ontario
Router#show snmp chassis
JAX123456789
Router#
```

It is also useful to extract this information from a backup of the router's configuration file. It is a good network management practice to keep a backup copy of every router's configuration on a central server such as the NMS. Then you can extract vital information such as the router's serial number. You will often need this information for service and support when a device fails and you can't reach it through SNMP. On a Unix server, you can use the *grep* utility to easily extract the required information:

```
Freebsd% grep snmp-server Router.confg
snmp-server community ORARO RO
snmp-server community ORARW RW
snmp-server location 999 Queen St. W., Toronto, Ontario
snmp-server contact Ian Brown 416-555-2943
snmp-server chassis-id JAX123456789
snmp-server host 172.25.1.1 ORATRAP
Freebsd%
```

If the router is reachable, you can also extract this information via SNMP:

```
Freebsd% snmpget -v1 -c ORARO Router .1.3.6.1.2.1.1.6.0
system.sysLocation.0 = 999 Queen St. W., Toronto, Ontario
Freebsd% snmpget -v1 -c ORARO Router .1.3.6.1.2.1.1.4.0
system.sysContact.0 = Ian Brown 416-555-2943
Freebsd% snmpget -v1 -c ORARO Router .1.3.6.1.4.1.9.3.6.3.0
enterprises.9.3.6.3.0 = "JAX123456789"
```

```
Freebsd%
```

Recipe 17.4 uses this information to construct a summary inventory report for all of the routers in a network.

17.3.4 See Also

Recipe 17.4

Top

Recipe 17.4 Extracting Inventory Information from a List of Routers with SNMP

17.4.1 Problem

You want to build a report of important router information for all of your managed routers.

17.4.2 Solution

The following Perl script extracts important router information such as router name, physical locations, contact names, and serial numbers from a list of routers, and creates a report of this information. The script is intended to be run manually and no arguments are required or expected.

Here's some example output:

```
Freebsd% ./inventory.pl
  Router              Location                      Contact              Serial
Router      999 Queen St. W., Toronto, Ont  Ian Brown 416-555-2943     JAX123456
Boston      1273 Main Street, Boston, MA    Bob Irwin 800-555-1221     JAX231567
Denver      24 Sussex Drive, Denver, CO     Helpdesk  800-555-2992     JAX928362
Frame       999 Queen St. W., Toronto, Ont  Ian Brown 416-555-2943     JAX212321
Toronto     999 Queen St. W., Toronto, Ont  Ian Brown 416-555-2943     JAX283291
Boston2     1273 Main Street, Boston, MA    Bob Irwin 800-555-1221     JAX292228
Denver2     24 Sussex Drive, Denver, CO     Helpdesk  800-555-2992     JAX219115
Freebsd%
```

Example 17-1 contains the Perl code.

Example 17-1. inventory.pl

```
#!/usr/bin/perl
#
#    inventory.pl -- a script to extract valuable information
#                    from a Router.  (Name, Location, Contact, S/N)
#
#
# Set behaviour
$workingdir="/home/nms";
$snmpro="ORARO";
$rtrlist="$workingdir/RTR_LIST";
#
#
open (RTR, "$rtrlist") || die "Can't open $rtrlist file";
```

```
open (LOG, ">$workingdir/RESULT") || die "Can't open $workingdir/RESULT file";
printf "  Router\t\t Location\t\t\tContact\t\t  Serial\n";
printf LOG  "  Router\t\t; Location\t\t\t;Contact\t\t  ;Serial\n";
while (<RTR>) {
  chomp($rtr="$_");
  $snmpget="/usr/local/bin/snmpget -v1 -c $snmpro $rtr ";
   $rtr=`$snmpget  .1.3.6.1.4.1.9.2.1.3.0`;
   $loc=`$snmpget  .1.3.6.1.2.1.1.6.0`;
   $con=`$snmpget  .1.3.6.1.2.1.1.4.0`;
   $sin=`$snmpget  .1.3.6.1.4.1.9.3.6.3.0`;
  chomp(($foo, $RTR) = split (/"/, $rtr));
  chomp(($foo, $LOC) = split (/= /, $loc));
  chomp(($foo, $CON) = split (/= /, $con));
  chomp(($foo, $SIN) = split (/"/, $sin));
   printf ("%-12.12s  %-30.30s  %-25.25s  %-12.12s\n", $RTR, $LOC, $CON, $SIN);
   printf LOG ("%-12.12s; %-30.30s; %-25.25s; %-12.12s\n", $RTR, $LOC, $CON, $SIN);
}
```

17.4.3 Discussion

This script extracts important router information from a list of routers and presents that list in a semicolon-delimited file, as well as displaying it on the screen. It is a very simple script that just extracts MIB variables from a list of routers using *snmpget*, illustrating the value and versatility of SNMP.

This Perl script works by cycling through a list of routers. For each router, it uses SNMP get requests to extract the values of four variables: router name, location, contact, and serial number, as we did manually in Recipe 17.3. The core of this script is NET-SNMP's *snmpget* utility. NET-SNMP must be present on the server and the script expects to find *snmpget* in the */usr/local/bin* directory.

This script contains three important variables that you have to set correctly before you can run this script. First, the variable *$workingdir* contains the directory in which the script resides. Next is the read-only SNMP community string, which is stored in the script's *$snmpro* variable. Substitute your organization's SNMP read-only community string for the example value shown (*ORARO*). Note that this script assumes that you use the same SNMP read-only community string on all routers.

The third variable, *$rtrlist*, contains the location of the router list. The example script uses a file called *RTR_LIST*, which is located in the working directory. You will need to change this variable to point to a file containing a list of routers on your network. The script expects this file to have a single router name per line.

The script produces two types of output. It displays the results on your screen while the script is executing. And the script also logs all results to a file called *RESULT* that resides in the working directory. The script automatically creates this file the first time you run it, and it overwrites the contents on each subsequent execution.

17.4.4 See Also

Recipe 17.3

Top

Recipe 17.5 Using Access Lists to Protect SNMP Access

17.5.1 Problem

You want to provide extra security to SNMP using access lists.

17.5.2 Solution

You can use the following commands to restrict which IP source addresses are allowed to access SNMP functions on the router. This is the legacy method:

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#access-list 99 permit 172.25.1.0 0.0.0.255
Router(config)#access-list 99 permit host 10.1.1.1
Router(config)#access-list 99 deny any
Router(config)#snmp-server community ORARO ro 99
Router(config)#access-list 98 permit 172.25.1.0 0.0.0.255
Router(config)#snmp-server community ORARW rw 98
Router(config)#end
Router#
```

Here is a newer method to do the same thing using SNMP server groups:

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#access-list 99 permit 172.25.1.0 0.0.0.255
Router(config)#access-list 99 permit host 10.1.1.1
Router(config)#access-list 99 deny any
Router(config)#snmp-server group COOKRO v1 access 99
Router(config)#snmp-server user TESTRO1 COOKRO v1
Router(config)#end
Router#
```

17.5.3 Discussion

By default, when you enable inbound SNMP services, the router will permit all IP addresses to access the SNMP agent on the standard well-known UDP port number (161). We highly recommend using SNMP ACLs to restrict SNMP access to a few trusted hosts or subnets. This will help to protect sensitive data.

You could restrict SNMP access by simply applying an interface ACL to block incoming SNMP

packets that don't come from trusted servers. However, this would not be as effective as using the global SNMP commands shown in this recipe. Because you can apply this method once for the whole router, it is much simpler than applying ACLs to block SNMP on all interfaces separately. Also, using interface ACLs would block not only SNMP packets intended for this router, but also would stop SNMP packets that just happened to be passing through on their way to some other destination device.

Despite the different syntax, both of the examples shown in this recipe work the same way. First, you define a standard access list. Then you apply this access list to your SNMP community string. You can assign each SNMP community a separate and unique access list to restrict access. This can be useful when there are several different people or NMS systems that need to access information on different groups of routers. You can also assign an access list to a read-write community string to block SNMP Set commands from unauthorized IP source addresses.

SNMP access lists alone are not an effective way of protecting read-write access to your routers. Because SNMP is a UDP protocol, a rogue device with access to your network can spoof the IP source address so that it matches the IP address of your management server. This means that somebody who knows your community string and the IP address of your management server can submit potentially dangerous SNMP Set commands to your router. SNMP sends packets in clear-text, so this information is relatively easy to discover with a protocol analyzer on a well-chosen network segment.

> Applying a nonexistent access list to your SNMP configuration commands implicitly allows all IP source addresses to access your router's SNMP services. Always ensure that the access list exists before applying it to an SNMP community string.

The following command shows the status of a particular access list:

```
Router#show access-list 99
Standard IP access list 99
     permit 10.1.1.1 (1745 matches)
     permit 172.25.1.0, wildcard bits 0.0.0.255 (477 matches)
    deny   any (7 matches)
Router#
```

Note that the router has denied seven requests. This means that the router received and discarded seven SNMP requests because source addresses were not allowed. Although the router automatically appends an implicit deny all to the end of every access list, we recommend that you explicitly include a deny all statement at the end of each access list to let you keep track of denied packets like this. It is an easy way to tell how often your routers receive SNMP requests from bad source addresses.

The *show snmp group* command shows you which access lists are assigned to which SNMP community strings:

```
Router>show snmp group
groupname: ORARO                          security model:v1
```

```
readview :v1default                          writeview: <no writeview specified>
notifyview: <no notifyview specified>
row status: active       access-list: 99

groupname: ORARW                             security model:v1
readview :v1default                          writeview: v1default
notifyview: <no notifyview specified>
row status: active       access-list: 98

Router>
```

In this example, the SNMP community string *ORARO* is protected by access list 99 and SNMP community string *ORARW* with access list 98. Note that the*show snmp group* command is only available in 12.0(3)T and above.

We discuss how to use the new SNMP syntax for read-write access in Recipe 17.7 and Recipe 17.21.

17.5.4 See Also

Recipe 17.1; Recipe 17.6; Recipe 17.7; Recipe 17.21; Chapter 19

Top

Recipe 17.6 Logging Unauthorized SNMP Attempts

17.6.1 Problem

You want to log unauthorized SNMP attempts.

17.6.2 Solution

Use the following commands to configure your router to log unauthorized SNMP requests:

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#access-list 99 permit 172.25.1.0 0.0.0.255
Router(config)#access-list 99 permit host 10.1.1.1
Router(config)#access-list 99 deny any log
Router(config)#snmp-server community ORARO ro 99
Router(config)#snmp-server community ORARW rw 99
Router(config)#end
Router#
```

17.6.3 Discussion

If you are concerned about unauthorized access to SNMP services on your router, it can be quite useful to configure the router to maintain detailed records of every failed request. These verbose log messages can provide information on incorrectly configured management servers, as well as malicious (or just plain nosy) users.

Simply adding the keyword *log* to the *deny any* line in your access list instructs the router to log all unauthorized SNMP attempts.

The following command will display the status of your SNMP access list:

```
Router#show access-list 99
Standard IP access list 99
    permit 10.1.1.1  (1293 matches)
    permit 172.25.1.0, wildcard bits 0.0.0.255 (630 matches)
    deny   any log (17 matches)
Router#
```

Unlike the example shown in Recipe 17.5, the *show access-list* output now includes the *log* keyword on the *deny any* line. The router will now send information on every unauthorized SNMP request to the

logging facility (see Chapter 18 for more information on logging). Use the *show logging* EXEC command to view the router's internal logging buffer:

```
Router#show logging
Syslog logging: enabled (0 messages dropped, 0 flushes, 0 overruns)
    Console logging: disabled
     Monitor logging: level debugging, 26 messages logged
        Logging to: vty2(0)
     Buffer logging: level debugging, 49 messages logged
     Trap logging: level informational, 53 message lines logged
        Logging to 172.25.1.1, 53 message lines logged
        Logging to 172.25.1.3, 53 message lines logged

Log Buffer (4096 bytes):
Apr 15 22:33:21: %SEC-6-IPACCESSLOGS: list 99 denied 192.168.22.13 1 packet
Apr 15 22:39:18: %SEC-6-IPACCESSLOGS: list 99 denied 10.121.212.11 3 packets
Router#
```

This example shows that access list 99, our SNMP access list, has denied access attempts by two IP source addresses, `192.168.22.13` and `10.121.212.11`. The final logging entry shows that the ACL denied three packets from source address `10.121.212.11`. Note that every packet received doesn't result in a separate log entry. If you are building a custom script to extract failed SNMP attempts, you will need to keep this in mind.

17.6.4 See Also

Recipe 17.5; Chapter 18

Top

Recipe 17.7 Limiting MIB Access

17.7.1 Problem

You want to limit which MIB variables can be remotely accessed with SNMP.

17.7.2 Solution

You can use the following commands to restrict SNMP access to portions of the MIB tree. This example shows the legacy configuration method:

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#access-list 99 permit 172.25.1.0 0.0.0.255
Router(config)#access-list 99 deny any log
Router(config)#snmp-server view ORAVIEW mib-2 included
Router(config)#snmp-server view ORAVIEW at excluded
Router(config)#snmp-server view ORAVIEW cisco included
Router(config)#snmp-server community ORARO view ORAVIEW ro 99
Router(config)#snmp-server view RESTRICTED lsystem.55  included
Router(config)#snmp-server community ORARW view RESTRICTED rw 99
Router(config)#end
Router#
```

Cisco also has a new method for restricting MIB access, which uses the *snmp-server group* command:

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#snmp-server view ORAVIEW mib-2 included
Router(config)#snmp-server view ORAVIEW at excluded
Router(config)#snmp-server view ORAVIEW cisco included
Router(config)#snmp-server group TEST v1 read ORAVIEW
Router(config)#snmp-server user ORARO TEST v1
Router(config)#snmp-server view RESTRICTED lsystem.55 included
Router(config)#snmp-server group TEST2 v1 write RESTRICTED
Router(config)#snmp-server user ORARW TEST2 v1
Router(config)#end
Router#
```

17.7.3 Discussion

By default, enabling SNMP services on your router allows SNMP servers to access the entire SNMP

MIB tree. However, you might want to limit which MIB variables can be remotely retrieved or changed, usually for security reasons. We strongly recommend that you limit SNMP write access to only those MIB objects that you absolutely need to change remotely. Remember that it is very easy for a malicious user to cause serious network problems by modifying MIB variables that control the router's configuration.

You can assign an SNMP MIB view to an individual community string or share a view among several community strings (including both read-only and read-write access strings). Assigning a MIB view to a read-only community string restricts which MIB variables can be displayed. Similarly, assigning an SNMP MIB view to a read-write community string restricts which MIB variables you can view or alter.

A MIB view can restrict access to a single MIB object, it can allow access to all but one MIB object, or anything in between. For instance, in both examples we created a view named *RESTRICTED* to the read-write community string *ORARW*. This view restricts access to a single MIB entry, *lsystem.55*, which is the MIB object that triggers the router to send its configuration file to a TFTP server (for nightly configuration backups). The router will prevent any other access to the MIB tree.

We also created an SNMP view named *ORAVIEW* which is less restrictive. In this case, we want to allow access to the MIB-2 variables but prevent access to the ARP table (AT) tree, which we can do by using the *exclude* keyword. At the same time, we allow access to the entire Cisco proprietary MIB tree by including the *cisco* MIB.

To illustrate the functionality of SNMP MIB views, here's an SNMP walk of a router's default MIB tree:

```
Freebsd% snmpwalk -v1 -c ORARO Router
system.sysDescr.0 = Cisco Internetwork Operating System Software
IOS (tm) C2600 Software (C2600-JK9O3S-M), Version 12.2(7a), RELEASE SOFTWARE (fc2)
Copyright (c) 1986-2002 by cisco Systems, Inc.
Compiled Thu 21-Feb-02 03:48 by pwade
system.sysObjectID.0 = OID: enterprises.9.1.209
system.sysUpTime.0 = Timeticks: (26809590) 3 days, 2:28:15.90
system.sysContact.0 = Ian Brown 416-555-2943
system.sysName.0 = Router.oreilly.com
system.sysLocation.0 = 999 Queen St. W., Toronto, Ont.
system.sysServices.0 = 78
system.sysORLastChange.0 = Timeticks: (0) 0:00:00.00
interfaces.ifNumber.0 = 10
interfaces.ifTable.ifEntry.ifIndex.1 = 1
interfaces.ifTable.ifEntry.ifIndex.2 = 2
interfaces.ifTable.ifEntry.ifIndex.3 = 3
interfaces.ifTable.ifEntry.ifIndex.4 = 4
interfaces.ifTable.ifEntry.ifIndex.5 = 5
interfaces.ifTable.ifEntry.ifIndex.6 = 6
interfaces.ifTable.ifEntry.ifIndex.7 = 7
interfaces.ifTable.ifEntry.ifIndex.8 = 8
interfaces.ifTable.ifEntry.ifIndex.9 = 9
```

```
<8000+ lines Removed>
End of MIB
Freebsd%
```

Walking the full MIB tree of a Cisco router can take a great deal of time. This router's MIB tree consisted of more than 8000 entries. However, if we implement a simple SNMP MIB view, the result is quite different:

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#snmp-server view TEST system.5 included
Router(config)#snmp-server community COOKBOOK view TEST ro
Router(config)#end
Router#
```

In this example, the router restricts access to a single MIB entry, *sysName* (system.5). Now when we attempt to walk the entire MIB tree again, the router sends only this single variable:

```
Freebsd% snmpwalk -v1 -c COOKBOOK Router
system.sysName.0 = Router.oreilly.com
End of MIB
Freebsd%
```

Note that the router displays a single entry, *sysName*, and reports that it has reached the "End of MIB," effectively preventing more than 8000 MIB objects from being accessed via this particular community string.

You can use the *show snmp group* EXEC command to see which views are assigned to which community string:

```
Router>show snmp group
groupname: ORARO                          security model:v1
readview :v1default                       writeview: <no writeview specified>
notifyview: <no notifyview specified>
row status: active

groupname: COOKBOOK                       security model:v1
readview :TEST                            writeview: <no writeview specified>
notifyview: <no notifyview specified>
row status: active

Router>
```

In this example, the community string *ORARO* has the default SNMP view, *v1default*. This means the entire MIB tree is accessible.

To see which MIB entries are assigned to which SNMP MIB view, use the following (hidden) command:

```
Router#show snmp view
```

```
ORAVIEW mib-2 - included nonvolatile active
ORAVIEW at - excluded nonvolatile active
ORAVIEW cisco - included nonvolatile active
v1default internet - included volatile active
v1default internet.6.3.15 - excluded volatile active
v1default internet.6.3.16 - excluded volatile active
v1default internet.6.3.18 - excluded volatile active
RESTRICTED cisco - included nonvolatile active
RESTRICTED lsystem.55 - included nonvolatile active
Router#
```

Table 17-2Table 17-2 lists a number of valid MIB trees that the router will accept within an SNMP view. Keep in mind that this is not an exhaustive list and that the router will also accept OIDs in their numerical format.

Table 17-3. Valid OID-trees for use with SNMP views

Keyword	Description
internet	Entire MIB tree
mib-2	Entire MIB-II tree
system	System branch of the MIB-II tree
interfaces	Interface branch of the MIB-II tree
at	ARP table branch of the MIB-II tree
ip	IP routing table branch of the MIB-II tree
icmp	ICMP statistics branch of the MIB-II tree
tcp	TCP statistics branch of the MIB-II tree
udp	UDP statistics branch of the MIB-II tree
transmission	Transmission statistics of the MIB-II tree
snmp	SNMP statistics branch of the MIB-II tree
ospf	OSPF MIB
bgp	BGP MIB
rmon	RMON MIB
cisco	Cisco's enterprise MIB tree
x25	X.25 MIB

Keyword	Description
ifEntry	Interface statistics MIB objects
lsystem	Cisco's system MIB

17.7.4 See Also

Recipe 17.1; Recipe 17.2

Top

Recipe 17.8 Using SNMP to Modify a Router's Running Configuration

17.8.1 Problem

You want to use SNMP to download or modify a router's configuration.

17.8.2 Solution

To upload or download a current copy of your router's configuration file to a TFTP server via SNMP, you have to first configure the router for read-write SNMP access:

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#snmp-server community ORARW rw
Router(config)#end
```

To download the current configuration file, you need to create an empty file on your TFTP server. In this case we assume a Unix server, but TFTP server software is available for essentially every popular operating system. Send an SNMP command to the router to trigger the TFTP download:

```
Freebsd% touch /tftpboot/router.cfg
Freebsd% chmod 666 /tftpboot/router.cfg
Freebsd% snmpset -v1 -c ORARW Router.1.3.6.1.4.1.9.2.1.55. 172.25.1.1 s router.cfg
enterprises.9.2.1.55.172.25.1.1 = "router.cfg"
Freebsd%
```

You can use SNMP to trigger the router to upload a configuration file from your TFTP server via SNMP as follows:

```
Freebsd% echo "no ip source-route" > /tftpboot/new.cfg
Freebsd% echo "end" >> /tftpboot/new.cfg
Freebsd% chmod 666 /tftpboot/new.cfg
Freebsd% snmpset -v1 -c ORARW Router.1.3.6.1.4.1.9.2.1.53. 172.25.1.1 s new.cfg
enterprises.9.2.1.53.172.25.1.1 = "new.cfg"
Freebsd% snmpset -v1 -c ORARW Router.1.3.6.1.4.1.9.2.1.54.0 i 1
enterprises.9.2.1.54.0 = 1
Freebsd%
```

17.8.3 Discussion

The ability to extract or modify your router's configuration via SNMP is powerful yet scary. These

examples illustrate the power of SNMP read-write access and the main reason we advocate SNMP security features. We highly recommend that you read recipe Recipe 17.11 before allowing open SNMP write access on your routers. In that recipe we demonstrates an effective way to mitigate unauthorized tampering with your router's configuration files.

This first example illustrates how to extract your router's running configuration file to a TFTP server using SNMP. Before a typical TFTP server will accept a file transfer, a world-writable file must exist. On a Unix platform, the *touch* command creates this file, while the *chmod* command ensures that it has the proper file attributes.

The *snmpset* command instructs the router to send its running configuration file to a particular file on a particular TFTP server:

```
snmpset -v1 -c ORARW Router .1.3.6.1.4.1.9.2.1.55. 172.25.1.1 s router.cfg
```

In this command, *Router* is the name (or IP address) of the router. The read-write SNMP community string is *ORARW*. The MIB OID value is actually in two parts. The first part, `.1.3.6.1.4.1.9.2.1.55`, is the OID value in the Cisco MIB extension that instructs the router to send its configuration file. The second part (`172.25.1.1`, in this case) is the IP address of your TFTP server, and *router.cfg* is the name of the file as it will appear on the TFTP server. The single letter *s* before the file name designates that the argument that follows will be a character string.

Extracting a router's configuration file like this is extremely useful. The Bourne shell script in Example 17-2 uses this method to extract and store the current configuration file from a Cisco router. The script automates the commands listed in the solution section to simplify the extraction of router configuration files. This script takes a single argument, the router name or IP address, and it stores the router configuration file in the */tftpboot* directory. The file will be the name of the router, with *.auto* appended to it (e.g., *router.auto*).

Example 17-2. conf

```
#!/bin/sh
#
#    conf -- A compact script to extract router configs to a
#            tftp server.
#
#
#  set behavior
snmprw="ORARW"
tftp="172.25.1.1"
#
#
router=$1
if [ "$router" = "" ]; then
echo "Usage: `basename $0` <hostname | ip address>" >&2 && exit 1
else
rm /tftpboot/$router-auto
```

```
touch /tftpboot/$router-auto
chmod 666 /tftpboot/$router-auto
snmpset="snmpset -v1 -c $snmprw $router "
$snmpset .1.3.6.1.4.1.9.2.1.55.$tftp s $router-auto
if [ -w /tftpboot/$router-auto -a -s /tftpboot/$router-auto ]; then
echo "Completed Successfully"
else
echo "Operation Failed"
fi
fi
```

Run this script as follows:

```
Freebsd% ./conf router
Completed Successfully
Freebsd%
```

This script assumes that NET-SNMP is on the server, and requires two variables to be set, *snmprw* and *tftp*. The *snmprw* variable contains the SNMP read-write community string of your organization and the *tftp* variable contains the IP address of your TFTP server.

The second example in the solution loads new configuration commands into a router. You must have a world-readable file containing these router configuration commands in your TFTP directory before you can upload anything. So in the example we have created a simple configuration file. We used echo commands to create the file, although in practice you should probably use a text editor to help limit the number of typing errors in your router's configuration. The last line in the configuration file should have the *end* command. This prevents the router from complaining about an unexpected end to the configuration file.

Note that when you upload a configuration file like this, the router merges the commands into its existing configuration, just as it does when you type the commands at the router's console.

There are two important differences between *snmpset* commands to upload or download a configuration file. The first is the different OID value. Be very careful that you get the right value here, because you don't want to accidentally upload an old configuration when you're trying to download. The second difference is that after uploading the configuration file, we issued another different *snmpset* command. This second command saves the configuration changes to NVRAM. This is the same as logging into the router and typing *write memory* or *copy running-config startup-config*.

17.8.4 See Also

Recipe 17.2; Recipe 17.5; Recipe 17.7; Appendix A

Top

Recipe 17.9 Using SNMP to Copy a New IOS Image

17.9.1 Problem

You want use SNMP to remotely upgrade a router's IOS.

17.9.2 Solution

Before you can upload or download the router's IOS image to a TFTP server, you have to set up a valid read-write SNMP community string:

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#snmp-server community ORARW rw
Router(config)#end
```

Then you can download a copy of your router's current IOS file to your TFTP server with the following Unix commands:

```
Freebsd% touch /tftpboot/c2600-jk9o3s-mz.122-7a.bin
Freebsd% chmod 666 /tftpboot/c2600-jk9o3s-mz.122-7a.bin
Freebsd% snmpset -v1 -c ORARW Router .1.3.6.1.4.1.9.2.10.9. 172.25.1.1 s c2600-jk9o3s-
mz.122-7a.bin
enterprises.9.2.10.9.172.25.1.1  =  "c2600-jk9o3s-mz.122-7a.bin"
Freebsd%
```

Use the following commands to upload an IOS file from your TFTP server to the router's flash memory:

```
Freebsd% chmod 666 /tftpboot/c2600-jk9o3s-mz.122-7a.bin
Freebsd% snmpset -v1 -c ORARW Router .1.3.6.1.4.1.9.2.10.6.0 i 1
enterprises.9.2.10.6.0 = 1
Freebsd% snmpset -v1 -c ORARW Router.1.3.6.1.4.1.9.2.10.12. 172.25.1.1 s c2600-jk9o3s-
mz.122-7a.bin
enterprises.9.2.10.12.172.25.1.1  =  "c2600-jk9o3s-mz.122-7a.bin"
Freebsd%
```

17.9.3 Discussion

The first example demonstrates how to use SNMP to force a router to download its IOS file to a TFTP server. Most TFTP servers will not accept an incoming transfer unless the destination file is world writable. On Unix computers, the *touch* command creates a file, while the *chmod* command gives it the proper file attributes.

This *snmpset* command instructs the router to use TFTP to copy its IOS file to a particular file on the specified server:

```
Freebsd% snmpset -v1 -c ORARW Router .1.3.6.1.4.1.9.2.10.9. 172.25.1.1 s c2600-jk9o3s-mz.122-7a.bin
```

In this case, *Router* is the router's name or IP address, and we use *ORARW* for the read-write SNMP community string. The OID value of the Cisco MIB variable that instructs the router to transfer its IOS image is `.1.3.6.1.4.1.9.2.10.9`. You concatenate the server's IP address (which is `172.25.1.1` in this example) to the end of the OID value. The last argument, *c2600-jk9o3s-mz.122-7a.bin* , is the name of the file as it will appear on the TFTP server.

This command is useful because it allows you to easily build a central library of all running IOS versions on your network. Then, if you have problems with a router and need to replace it, you can easily make sure that the new device is running the same IOS version as the old one. It is also useful if you discover that a particular IOS version behaves better and want to copy that version into other routers.

The second example shows how to use SNMP to start a TFTP upload of a new IOS version. This is useful because it makes it easy to build a script to automate changing the IOS versions on a large number of similar routers. You should be careful when doing this, however. It is safe to copy a new IOS image into the router's flash memory on most Cisco routers. The router will continue running the old version until it reboots. A good procedure for doing a large number of upgrades like this is to run a script to copy the new images to the routers, then check each router to ensure that the upload completed successfully before rebooting.

The method shown for uploading a new IOS file to a router is similar to the method for downloading an IOS file. The one important difference is the step that erases the flash before the uploading commences. In our example, the first *snmpset* command erases the flash:

```
Freebsd% snmpset -v1 -c ORARW Router .1.3.6.1.4.1.9.2.10.6.0 i 1
enterprises.9.2.10.6.0 = 1
```

This step is necessary only if there is not enough flash space available to load the new IOS file. The second command actually copies the image into the router's flash memory:

```
Freebsd% snmpset -v1 -c ORARW Router.1.3.6.1.4.1.9.2.10.12. 172.25.1.1 s c2600-jk9o3s-mz.122-7a.bin
```

Note that some types of Cisco routers do not support this method for uploading IOS images. In particular, Cisco 2500 series routers actually run directly from the IOS image in flash instead of copying an image of the IOS into processor memory at boot time. Changing the IOS version in flash would cause serious problems, so the router will not allow you to do it.

17.9.4 See Also

Recipe 17.2 ; Recipe 17.5 ; Recipe 17.7 ; Chapter 1 ; Appendix A

Top

Recipe 17.10 Using SNMP to Perform Mass Configuration Changes

17.10.1 Problem

You want to automate the distribution of a set of configuration commands to a large number of routers.

17.10.2 Solution

The Perl script in Example 17-3 will distribute configuration commands to a large number of routers. It works using SNMP to trigger TFTP file transfers into the routers. In effect, this script lets you automatically distribute a series of configuration commands to a list of routers. Automating routine changes like this saves time and effort but more importantly, it virtually eliminates typing mistakes.

Here's some example output:

```
Freebsd% ./snmpcfg.pl
= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
toronto - Update Successful
toronto - Wr Mem Successful
= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
boston  - Update Successful
boston  - Wr Mem Successful
= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
denver  - Update Successful
denver  - Wr Mem Successful
= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
newyork - Update Successful
newyork - Wr Mem Successful
= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
detroit - Update Failed
= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
chicago - Update Successful
chicago - Wr Mem Successful
= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
sanfran - Update Successful
sanfran - Wr Mem Successful
= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
seattle - Update Successful
seattle - Wr Mem Successful
= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
Freebsd%
```

Example 17-3 contains the Perl code.

Example 17-3. snmpcfg.pl

```
#!/usr/bin/perl -w
#
#    snmpcfg.pl -- a script to perform mass configuration changes to
#                  a list of routers using SNMP.
#
#
# Set behaviour
$workingdir="/home/nms";
$snmprw="ORARW";
$tftpsrv="172.25.1.1";
#
#
$rtrlist="$workingdir/RTR_LIST";
open (RTR, "$rtrlist") || die "Can't open $rtrlist file";
open (LOG, ">$workingdir/RESULT") || die "Can't open $workingdir/RESULT file";
#
while (<RTR>) {
  chomp($rtr="$_");
  print LOG "= = = = = = = = = = = = = = = = = = = = = = = = = = = = = \n";
  print "= = = = = = = = = = = = = = = = = = = = = = = = = = = = = \n";
  $snmpset="/usr/local/bin/snmpset -t 20 -r 2 -v1 -c $snmprw $rtr ";
   chomp($result=`$snmpset  .1.3.6.1.4.1.9.2.1.50.$tftpsrv  s  SNMPCFG`);
  if ($result=~/.+ = "(.+)"$/ ) {
            if( $1 eq SNMPCFG ) {
                print LOG "$rtr - Update Successful\n";
                print "$rtr - Update Successful\n";
                chomp($result=`$snmpset .1.3.6.1.4.1.9.2.1.54.0 i 1`);
                if ($result=~/.+ = (.+)$/ ) {
                          if( $1 = = 1 ) {
                            print LOG "$rtr - Wr Mem Successful\n";
                            print "$rtr - Wr Mem Successful\n";
                          }
                          else {
                            print LOG "$rtr - Wr Mem Failed\n";
                            print "$rtr - Wr Mem Failed\n";
                          }
                }
                else {
                  print LOG "$rtr - Wr Mem Failed\n";
                  print "$rtr - Wr Mem Failed\n";
                }
            }
            else {
              print LOG "$rtr - Update Failed\n";
              print "$rtr - Update Failed\n";
            }
  }
```

```
  else {
    print LOG "$rtr - Update Failed\n";
    print "$rtr - Update Failed\n";
  }
}
```

17.10.3 Discussion

This script distributes a set of configuration commands to a list of routers using SNMP to trigger TFTP transfers, as we did manually in Recipe 17.8. The script goes through a list of routers in sequence, performing an *snmpset* command on each one to force the router to upload a predefined configuration file. If the file transfer completes successfully, the script will issue another *snmpset* command that permanently saves the running configuration file to NVRAM. The script displays a status report to the terminal screen and sends the same messages to a flat log file.

This script requires the NET-SNMP toolset. The script looks for the executable *snmpset* in the default location, */usr/local/bin*. If your system has *snmpset* in another location, change the variable *$snmpset*.

Before running the script, change the variable *$workingdir* to point to the directory where the script resides. Also set the variable *$snmprw* to your organization's SNMP read-write community string. This script will not work with a read-only community string. You will need to set the value of *$tftpsrv* to the IP address of the TFTP server where the configuration file resides.

The script expects to find the router list located in the working directory in a file called *RTR_LIST*. This file should have a single router name per line. You can change the default name and location of this file by modifying the variable *$rtrlist*.

By default, the script will copy the configuration file *SNMPCFG* (which is located in the */tftpboot* directory) to every router in the list. The configuration file must be world readable. This file should include a list of Cisco configuration commands as you would type them from a command prompt on the router. As we discussed in Chapter 1, we recommend inserting the keyword *end* at the end of the configuration file to prevent spurious error messages. If you want to change the filename, you will need to change both occurrences of the default filename *SNMPCFG* to the name of the new file.

The script creates a status report in a file called *RESULT* in the working directory. The script will automatically create this file the first time you execute it and will clear its contents each time the script is run. The status file allows you to run the script unattended and check for failures later. The easiest way to check for failures is to use the Unix *grep* utility to search the status report file for the keyword *Fail*.

17.10.4 See Also

Recipe 17.2; Recipe 17.5; Recipe 17.7; Recipe 17.8; Chapter 1; Appendix A

Top

Recipe 17.11 Preventing Unauthorized Configuration Modifications

17.11.1 Problem

You want to ensure that only authorized devices can use SNMP and TFTP to send or receive configuration information.

17.11.2 Solution

You can use the *snmp-server tftp-server-list* configuration command to restrict which TFTP servers the router can use in response to an SNMP trigger to upload or download configuration information:

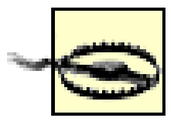
```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#access-list 92 permit 172.25.1.1
Router(config)#access-list 92 deny any log
Router(config)#snmp-server tftp-server-list 92
Router(config)#snmp-server community ORARW rw
Router(config)#end
Router#
```

17.11.3 Discussion

By default, the router will send or receive configuration information to any TFTP server. But this can be dangerous because the SNMP request that triggers these transfers cannot be 100% protected. Recipe 17.5 showed how you can restrict SNMP access to a specified list of devices. But, because SNMP uses UDP, it is not difficult for a malicious user to put the IP address of one of these allowed devices in the source of an SNMP packet, which will cause the router to execute the request. This packet could instruct the router to upload or download configuration information to or from any TFTP server. The attacker could then easily compromise the security of the entire network.

Therefore, we strongly recommend that you use the *tftp-server-list* command to restrict the TFTP servers to which your router will forward its configuration file and the TFTP servers from which your router will accept configuration changes.

It is important to note that this command restricts only TFTP sessions that the router initiates via SNMP. You can still use other TFTP servers for file transfers initiated from the router's command prompt.



If the access list assigned to the *tftp-server-list* does not exist, the router implicitly allows access for all TFTP servers.

The example authorizes the router to access only a single TFTP server. Note that the access list is designed to log all unauthorized attempts:

```
Router(config)#access-list 92 permit 172.25.1.1
Router(config)#access-list 92 deny any log
```

We highly recommend doing this because it not only prevents unauthorized access, but it also gives you information about what devices have been involved in the attempts. If there are malicious users with access to your network, this can help you to figure out who they are.

Note that this is a global command that affects all SNMP read-write community strings. There is no way to specify a different *tftp-server-list* for each community string.

17.11.4 See Also

Recipe 17.1; Recipe 17.5

Top

Recipe 17.12 Making Interface Table Numbers Permanent

17.12.1 Problem

You want to ensure that your router uses the same SNMP interface numbers every time it reboots.

17.12.2 Solution

To ensure that SNMP interface numbers remain permanent after a router power cycle, use the following command. This is a global command that affects all interfaces:

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#snmp-server ifindex persist
Router(config)#end
Router#
```

You can also fix the SNMP interface number of a single interface:

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#interface Serial0/0
Router(config-if)#snmp ifindex persist
Router(config-if)#end
Router#
```

This command is available in IOS Versions 12.1(5)T and above.

17.12.3 Discussion

It comes as a surprise to most engineers that the internal SNMP interface numbers assigned by the router are not stable. The SNMP interface numbers are prone to change after router reboot, especially if you add or remove logical interfaces (i.e., subinterfaces) or physical modules.

This issue has plagued many administrators and software vendors for years. The problem is that most network performance software packages poll for interface data using the unique interface number assigned by the router. However, if these numbers change after a router reboots, then the performance data becomes meaningless because there is no guarantee that you are still polling the same interface. Most high-end SNMP performance software companies have built fixes to circumvent this exact issue.

Changing interface numbers particularly affects the router's built-in RMON monitoring. With RMON you can configure the router to monitor its own MIB values and assign threshold values for sending notifications. Unfortunately, before the new functionality shown in this recipe came along, RMON polling was not reliable for interface-specific statistics. RMON services are discussed in detail in Recipe 17.22.

There are some minor costs to using this feature. First, each interface number requires 25 bytes of NVRAM to store. Second, some administrators have reported slightly slower boot times on routers that employ this feature. Otherwise, this new functionality is mostly transparent to the network administrator.

To illustrate the *ifindex* stability problem, consider the interface numbers on a typical router:

```
Freebsd% snmpwalk -v1 -c ORARO Router ifDescr
interfaces.ifTable.ifEntry.ifDescr.1 = "BRI0/0"
interfaces.ifTable.ifEntry.ifDescr.2 = "Ethernet0/0"
interfaces.ifTable.ifEntry.ifDescr.3 = "BRI0/0:1"
interfaces.ifTable.ifEntry.ifDescr.4 = "BRI0/0:2"
interfaces.ifTable.ifEntry.ifDescr.5 = "FastEthernet1/0"
interfaces.ifTable.ifEntry.ifDescr.6 = "Null0"
interfaces.ifTable.ifEntry.ifDescr.7 = "Loopback0"
```

Note that the router assigns a unique number to each interface, starting with 1. In this example, the interface `FastEthernet1/0` has an *ifindex* value of 5. This is the number you would use in SNMP polls for various interface level performance statistics. Next, we will power down the router and remove the BRI module before restoring power:

```
Freebsd% snmpwalk -v1 -c ORARO Router ifDescr
interfaces.ifTable.ifEntry.ifDescr.1 = "Ethernet0/0"
interfaces.ifTable.ifEntry.ifDescr.2 = "FastEthernet1/0"
interfaces.ifTable.ifEntry.ifDescr.3 = "Null0"
interfaces.ifTable.ifEntry.ifDescr.4 = "Loopback0"
```

Note that the BRI interface entries are gone and the remaining interface numbers have completely changed. The `FastEthernet1/0` interface now appears as interface number 2. And, worse still, there is no interface number 5 at all. So if you had been doing performance analysis on this port, it would suddenly stop working.

Returning the router to its original state restores the original interface numbers:

```
Freebsd% snmpwalk -v1 -c ORARO Router ifDescr
interfaces.ifTable.ifEntry.ifDescr.1 = "BRI0/0"
interfaces.ifTable.ifEntry.ifDescr.2 = "Ethernet0/0"
interfaces.ifTable.ifEntry.ifDescr.3 = "BRI0/0:1"
interfaces.ifTable.ifEntry.ifDescr.4 = "BRI0/0:2"
interfaces.ifTable.ifEntry.ifDescr.5 = "FastEthernet1/0"
interfaces.ifTable.ifEntry.ifDescr.6 = "Null0"
interfaces.ifTable.ifEntry.ifDescr.7 = "Loopback0"
```

However, if we enable the *snmp ifindex persist* command before powering down the router and removing the BRI module, the only difference is that the three entries associated with the BRI interface are removed:

```
Freebsd% snmpwalk -v1 -c ORARO 172.25.1.8 ifDescr
interfaces.ifTable.ifEntry.ifDescr.2 = "Ethernet0/0"
interfaces.ifTable.ifEntry.ifDescr.5 = "FastEthernet1/0"
interfaces.ifTable.ifEntry.ifDescr.6 = "Null0"
interfaces.ifTable.ifEntry.ifDescr.7 = "Loopback0"
```

The remaining interfaces have retained their original interface numbers after the router reboot. In particular, the `FastEthernet1/0` interface is once again interface number 5, which means that all polled data will still be useful.

17.12.4 See Also

Recipe 17.22

Top

Recipe 17.13 Enabling SNMP Traps and Informs

17.13.1 Problem

You want the router to generate SNMP traps or informs in response to various network events.

17.13.2 Solution

The following configuration commands will enable your router to send unsolicited SNMP traps to a network management server:

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#snmp-server enable traps
Router(config)#snmp-server host 172.25.1.1 ORATRAP  config entity envmon hsrp
Router(config)#snmp-server host nms.oreilly.com ORATRAP bgp snmp envmon
Router(config)#end
Router#
```

Notice that the *snmp-server host* command will accept either an IP address or a hostname.

Beginning with SNMPv2c, Cisco routers also support SNMP informs. To enable SNMP informs, use the following commands:

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#snmp-server enable informs
Router(config)#snmp-server host 172.25.1.1 informs version 2c  ORATRAP snmp envmon
Router(config)#end
Router#
```

17.13.3 Discussion

SNMP traps originate from the router's agent and are sent via UDP (port 162) to the Network Management Station (NMS). Unlike the information that the router sends to the NMS in response to an SNMP poll, a trap is unsolicited. The router's agent decides that something important has happened, and that it needs to tell the NMS about it. You must enable global trap support (see Table 17-3Table 17-3) and configure the trap host before the router agent can send traps.

SNMP traps are one of the basic elements of fault management. In fact, RFC 1812 ("Requirements for IP

Version 4 Routers") states that all routers must be capable of sending SNMP traps.

Cisco routers can send a large variety of different SNMP traps, including both standard traps described in RFCs, and Cisco-specific traps. The first step in configuring trap support is to enable the particular trap types you wish to use. In our examples, we choose to enable all SNMP trap types using the configuration command *snmp-server enable traps*. The fact that we don't specify individual trap types implicitly enables all trap types. However, you can restrict the router to send only certain types of traps that you are interested in receiving. The various trap type keywords are shown in Table 17-3Table 17-3. Note that this is a global command that affects all SNMP trap receivers.

Table 17-4. Cisco SNMP trap types

Keyword	Description
bgp	Allow BGP state change traps
bstun	Allow bstun event traps
config	Allow SNMP configuration traps
dlsw	Allow DLSw traps
dsp	Allow SNMP DSP traps
dspu	Allow dspu event traps
entity	Allow SNMP entity traps
envmon	Allow environmental monitor traps
frame-relay	Allow SNMP Frame Relay traps
hsrp	Allow SNMP HSRP traps
ipmulticast	Allow SNMP IP multicast traps
isdn	Allow SNMP ISDN traps
msdp	Allow SNMP MSDP traps
rsrb	Allow rsrb event traps
rsvp	Allow RSVP flow change traps
rtr	Allow SNMP Response Time Reporter (RTR) traps
sdlc	Allow SDLC event traps
sdllc	Allow SDLLC event traps

Keyword	Description
snmp	Allow SNMP-type notifications
stun	Allow stun event traps
syslog	Allow SNMP syslog traps
tty	Allow TCP connection traps
udp-port	The server host's UDP port number
voice	Allow SNMP voice traps
x25	Allow X25 event traps
xgcp	Allow XGCP protocol traps

For example, you could use the following commands to tell the router to send only BGP and environmental type traps:

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#snmp-server enable traps bgp
Router(config)#snmp-server enable traps envmon
Router(config)#end
Router#
```

You can also disable a particular type of SNMP trap with the following command:

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#no snmp-server enable traps envmon
Router(config)#end
Router#
```

The following command displays which SNMP trap types are enabled on a router:

```
Router#show running-config | include snmp-server enable
snmp-server enable traps snmp authentication linkdown linkup coldstart warmstart
snmp-server enable traps hsrp
snmp-server enable traps config
snmp-server enable traps entity
snmp-server enable traps envmon
snmp-server enable traps bgp
snmp-server enable traps ipmulticast
snmp-server enable traps msdp
snmp-server enable traps rsvp
snmp-server enable traps frame-relay
snmp-server enable traps syslog
snmp-server enable traps rtr
snmp-server enable traps dlsw
```

```
snmp-server enable traps dial
snmp-server enable traps dsp card-status
snmp-server enable traps voice poor-qov
Router#
```

The second step in configuring SNMP traps is to define the trap recipient using the *snmp-server host* command. This command has the following attributes:

snmp-server host host-addr [traps | informs] [version {1 | 2c}] community-string [udp-port *port] [trap-type* **]**

The *host-addr* argument is the name or IP address of the NMS server that will receive the traps. You can define whether the router will send SNMP traps or informs to this host by specifying either the *traps* or *informs* keyword. If neither is specified, the default is to send traps. Also, you can specify which version of SNMP traps the router will send by including either *version 1* or *version 2c*. If neither version is specified, the router will default to Version 1. Note that informs don't exist in SNMP Version 1, so you must specify Version 2c (or Version 3) if you want to enable this feature.

The *community-string* argument specifies the community string that the router will send within the SNMP trap or inform. This doesn't need to match either the read-only or read-write community strings on the router.

You can change the default SNMP trap port from 162 (the default) to another value with the optional *udp-port* keyword. The alternative UDP port number that you want to use must follow this keyword.

Finally, if the *trap-type* keyword is present, it allows you to configure the types of traps that the router will send to this server. The command can accept one or more types. However, if no trap types are included, the router will default to sending every enabled trap type.

There are two important things to note about this command. First, you must enable trap types via the global command before you can specify them for a particular host. Second, this command will allow you to send different sets of traps to different servers. This can sometimes be useful if you have multiple NMS servers that handle different management functions.

The configuration for SNMP informs is almost the same as that for SNMP traps. The main difference is that you can't enable individual inform types using the global *snmp-server enable informs* command. The global inform command lacks the granularity of the same trap-based command. However, you can still enable specific inform types on the host-level command. This can mean more typing if there are several inform recipients, but no functionality is lost.

17.13.4 See Also

Recipe 17.21; RFC 1812

Top

Recipe 17.14 Sending syslog Messages as SNMP Traps and Informs

17.14.1 Problem

You want to send syslog messages as SNMP traps or informs.

17.14.2 Solution

You can configure the router to forward syslog messages to your network management server as SNMP traps instead of syslog packets with the following configuration commands:

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#logging history informational
Router(config)#snmp-server enable traps syslog
Router(config)#snmp-server host 172.25.1.1 ORATRAP  syslog
Router(config)#end
Router#
```

To forward syslog messages as SNMP informs, use the snmp-server configuration commands:

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#logging history informational
Router(config)#snmp-server enable informs
Router(config)#snmp-server host 172.25.1.1 informs version 2c  ORATRAP syslog
Router(config)#end
Router#
```

17.14.3 Discussion

Cisco routers normally forward syslog messages via the syslog facility using UDP port 514. However, in networks that support SNMP traffic only, Cisco routers can encapsulate their syslog messages into SNMP traps before sending them.

This feature is most useful if your network management software doesn't support the syslog protocol. However, since routers can produce many more syslog messages than SNMP traps, we recommend using syslog where possible. Further, the fact that all of the syslog messages sent as SNMP traps use the same OID number can make parsing for particular log messages quite difficult.

Here is an example log message as it appears in the router's log:

```
Router#clear counters
Clear "show interface" counters on all interfaces [confirm]
```

```
Router#
May 28 10:07:04: %CLEAR-5-COUNTERS: Clear counter on all interfaces by ijbrown on
vty0 (172.25.1.1)
```

The router sends this message as a trap to the network management server, which records it in its trap log:

```
Freebsd% tail snmptrapd.log
May 28 10:07:04 freebsd snmptrapd[77759]: 172.25.25.1: Enterprise Specific Trap (1)
Uptime: 18 days, 22:35:26.99, enterprises.9.9.41.1.2.3.1.2 .118 = "CLEAR",
enterprises.9.9.41.1.2.3.1.3.118 = 6, enterprises.9.9.41.1.2.3.1.4.118 = "COUNTERS",
enterprises.9.9.41.1.2.3.1.5.118  = "Clear counter on all interfaces by ijbrown on
vty0 (172.25.1.1)", enterprises.9.9.41.1.2.3.1.6.118 = Timeticks: (163652698) 18
days, 22:35:26.98
Freebsd%
```

In this example, we forced the router to create a log message by clearing the interface counters. The router displayed the raw syslog message to the VTY session. The same information appears in the server's *snmptrapd.log* file. This is a flat file that NET-SNMP uses to store all SNMP traps forwarded to the server. Other network management systems store trap information in different formats and with different filenames.

You can also configure the router to forward syslog messages as SNMP informs. The result is the same as for traps. For more information on syslog (and logging in general), refer to Chapter 18

17.14.4 See Also

Recipe 17.13 ; Chapter 18

Top

Recipe 17.15 Setting SNMP Packet Size

17.15.1 Problem

You want to change the default SNMP packet size.

17.15.2 Solution

The following configuration command adjusts the default packet size for all SNMP packets leaving the router:

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#snmp-server packetsize 1480
Router(config)#end
Router#
```

17.15.3 Discussion

By default, Cisco routers limit their SNMP packet size to 1500 bytes. It is usually not necessary to change this parameter. However, it may be useful to reduce it if your network has an MTU of less than 1500 bytes, to prevent unnecessary fragmentation. The conventional wisdom on UDP packets is that smaller is usually better than larger because of fragmentation.

Conversely, if your network media can accepts a larger MTU than 1500 bytes, increasing your SNMP packet size can improve performance, particularly when transferring large MIB tables.

Note that adjusting the maximum SNMP packet size will affect all types of SNMP packets, including responses to SNMP get or set requests, as well as SNMP traps and SNMP informs. You can set the SNMP packet size to any integer between 484 and 17,940 bytes.

Top

Recipe 17.16 Setting SNMP Queue Size

17.16.1 Problem

You want to increase the size of a router's SNMP trap queue.

17.16.2 Solution

The following command increases the size of a router's SNMP trap queue:

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#snmp-server queue-length 25
Router(config)#end
Router#
```

To increase the size of the router's SNMP inform queue, use the following configuration command:

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#snmp-server inform pending 40
Router(config)#end
Router#
```

17.16.3 Discussion

By default, the router can hold 10 trap messages in its queue. The queue holds traps until the router can forward them to the NMS. The queue fills when the router generates traps faster than it can forward them. If it generates additional traps when the queue is already full, these new trap messages are dropped. The router has only one SNMP message queue for all trap recipients.

Regardless of the network's capacity, the router will never send SNMP messages faster than four traps per second. This rate is hardcoded into the router and is not configurable. If you have several NMS systems, or if your router creates a particularly large number of traps, you might want to increase the size of this queue to help prevent inadvertent discarding of traps.

The *snmp-server queue-length* command will accept any integer between 1 and 1000, representing the maximum number of packets that can be held at a time.

To show the current number of SNMP messages in the queue, and the maximum queue size, use the

show snmp EXEC command:

```
Router#show snmp
Chassis:  JAX123456789
Contact: Ian Brown 416-555-2943
Location: 999 Queen St. W., Toronto, Ontario
270 SNMP packets input
    0 Bad SNMP version errors
    12 Unknown community name
    0 Illegal operation for community name supplied
    0 Encoding errors
    231 Number of requested variables
    25 Number of altered variables
    11 Get-request PDUs
    222 Get-next PDUs
    25 Set-request PDUs
584 SNMP packets output
    0 Too big errors (Maximum packet size 1480)
    2 No such name errors
    0 Bad values errors
    0 General errors
    258 Response PDUs
    326 Trap PDUs

SNMP logging: enabled
    Logging to 172.25.1.1.162, 0/25, 309 sent, 17 dropped.
Router#
```

In this example, the `0/25` in the last line means that no SNMP traps are currently queued for transmission, and the queue can accept up to 25 messages at once. If you see the number of dropped SNMP traps growing rapidly over time, you should consider increasing the queue size. In this case, the "Trap PDUs" line tells you that the router has tried to send 326 traps. Of these, the last line tells you that it has successfully sent 309 and dropped 17. This is a 5% drop rate, which suggests that the queue depth should probably be increased.

SNMP informs maintain a queue for messages pending acknowledgement. Each inform is held in the pending queue until an acknowledgement is received. Consequently, the inform queue must be considerably larger than the corresponding trap queue.

If you choose to implement SNMP informs, we highly recommend that you increase the size of the pending queue from the default value of 25:

```
Router(config)#snmp-server inform pending 40
```

The router will accept any integer between 1 and 4,294,967,295 representing the number of unacknowledged informs to hold. There is very little benefit to using informs if the size of the pending queue is too small, because the router will not be able to hold new messages.

Top

Recipe 17.17 Setting SNMP Timeout Values

17.17.1 Problem

You want to adjust the SNMP trap timeout value.

17.17.2 Solution

You can use the following configuration command to adjust a router's SNMP trap timeout value:

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#snmp-server trap-timeout 60
Router(config)#end
Router#
```

To adjust a router's SNMP inform timeout value, use the following configuration commands:

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#snmp-server inform timeout 120
Router(config)#end
Router#
```

17.17.3 Discussion

Before a router can send an SNMP trap, it must have a route to the destination address of the trap. If a route to the SNMP server does not exist, the router will store the trap in the retransmission queue. By default, the router will hold a trap in the retransmission queue for 30 seconds before attempting to deliver it again. Sometimes it is useful to modify the default wait time to improve the chances of successful delivery.

For instance, if the router has to send the trap over a low-speed dial backup interface, 30 seconds may not be enough time for it to trigger a call, establish connectivity, and stabilize a routing table. In a situation like this, you should consider increasing the trap timeout. The value is specified as an integer number of seconds between 1 and 1000.

SNMP informs use timeouts differently. The inform timeout is the number of seconds that the router will wait for an acknowledgement before resending. The default value is 30 seconds but the router will accept any value between 0 and 4,294,967,295 seconds.

Note that increasing the timeout values for traps and informs means that the router will tend to hold these messages for longer. This, in turn, generally means that you will have to increase the queue size in order to hold them.

Top

Recipe 17.18 Disabling Link Up/Down Traps per Interface

17.18.1 Problem

You want to disable link up/down traps for specific interfaces.

17.18.2 Solution

To disable SNMP link status-change traps for a particular interface, use the following configuration command:

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#interface Serial0/0
Router(config-if)#no snmp trap link-status
Router(config-if)#end
Router#
```

17.18.3 Discussion

By default, the router forwards SNMP link up or down traps whenever an interface changes states. Normally, you want to receive traps when an interface changes states, because that could indicate a serious problem. But there are times when it is useful to disable these types of traps. For instance, dial interfaces may cycle up and down throughout the day without cause for concern. In these cases, you will probably want to suppress these types of traps to prevent network management staff from needlessly chasing meaningless failure reports.

It is also useful to disable SNMP traps for link up and down messages when you are troubleshooting, testing, or enabling interfaces to prevent extraneous failure reports.

This command will work only on physical interfaces and loopback interfaces. It will not allow you to disable link status traps on subinterfaces.

17.18.4 See Also

Recipe 17.13

Top

Recipe 17.19 Setting the IP Source Address for SNMP Traps

17.19.1 Problem

You want to set the source IP address for all SNMP traps leaving a router.

17.19.2 Solution

To set the default IP source address for all traps leaving a router, use the following configuration commands:

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#snmp-server host 172.25.1.1 ORATRAP
Router(config)#snmp-server trap-source loopback0
Router(config)#end
Router#
```

17.19.3 Discussion

Normally, when you enable SNMP traps to a remote server, that server will see the source IP address of the router's closest interface. However, this is not always meaningful. For instance, it is relatively common practice to populate your DNS with only the router's loopback interfaces. In this case, the server will not be able to resolve the originator of the trap.

Further, it can be difficult to correlate traps from the same router delivered through different interfaces. For example, this could happen as a result of a network failure. It can be confusing to see a link down message coming from one IP address and the corresponding link up message from a different one.

By enabling the *snmp-server trap-source* command, you can force the router to always use the same IP source address for all of the SNMP traps it sends. Industry best practices dictate that a loopback interface is usually the best choice for this because it is a virtual interface that is always available. Physical interfaces such as Ethernet or serial interfaces can become unavailable and limit the usefulness of this command. If you set the source interface to an unreachable interface, the router will resort to using the closest interface as the source address.

Note that Cisco's IOS will even allow you to assign a trap source interface without having an IP assigned address to it. However, the router will forward a syslog message highlighting the issue and resort to the default method of using the closest interface address for sending traps. Here is an example

of the log message that appears in this case:

```
 Jun 12 00:22:00 EDT: %IP_SNMP-4-NOTRAPIP: SNMP trap source Loopback1 has no ip
address
```

There is not yet an equivalent command for SNMP informs.

Top

Recipe 17.20 Using RMON to Send Traps

17.20.1 Problem

You want the router to send a trap when the CPU rises above a threshold, or when other important events occur.

17.20.2 Solution

You can configure a router to monitor its own CPU utilization and trigger an SNMP trap when the value exceeds a defined threshold with the following set of configuration commands:

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#rmon event 1 log trap  ORATRAP description "CPU on Router has exceeded
threshold" owner ijbrown
Router(config)#rmon event 2 log description " CPU on Router has normalized" owner
ijbrown
Router(config)#rmon alarm 1 lsystem.57.0 60 absolute rising-threshold 70 1 falling-
threshold 40 2 owner ijbrown
Router(config)#end
Router#
```

The following commands will configure the router to monitor its own buffer failures and send an SNMP trap when the number of failures exceeds a threshold:

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#rmon event 3 log trap  ORATRAP description "Excessive buffer failures
on Router" owner  ijbrown
Router(config)#rmon alarm 2 lsystem.46.0 300 delta rising-threshold  5 3 falling-
threshold -1 3 owner  ijbrown
Router(config)#end
Router#
```

To configure a router to monitor its own memory utilization and trigger an SNMP trap when it exceeds its threshold, use the following set of configuration commands:

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#rmon event 4 log trap  ORATRAP description "Low memory condition on
Router" owner ijbrown
Router(config)#rmon event 5 log trap  ORATRAP description "Low Memory condition
cleared on Router" owner ijbrown
Router(config)#rmon alarm 3 lsystem.8.0 60 absolute rising-threshold  1500000 5
```

```
falling-threshold 1000000 4 owner ijbrown
Router(config)#end
Router#
```

In this example, the router is configured to monitor the link utilization of a single interface and trigger an SNMP trap when that interface exceeds its threshold:

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#rmon event 6 log trap  ORATRAP description "Bandwidth utilization has
exceeded threshold on Router interface Serial 0/0 " owner ijbrown
Router(config)#rmon event 7 log trap  ORATRAP description "Bandwidth utilization has
normalized on Router interface Serial 0/0 " owner ijbrown
Router(config)#! Configure inbound alarm on Serial0/0 (ifNumber 3)
Router(config)#rmon alarm 4 lifEntry.6. 3 300 absolute rising-threshold  1000000 6
falling-threshold 800000 7 owner ijbrown
Router(config)#! Configure outbound alarm on Serial0/0 (ifNumber 3)
Router(config)#rmon alarm 5 lifEntry.8. 3 300 absolute rising-threshold  1000000 6
falling-threshold800000 7 owner ijbrown
Router(config)#end
Router#
```

To configure a router to monitor the number of interface errors on a particular interface, use the following set of commands:

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#rmon event 7 log trap  ORATRAP description "ifErrors have exceeded
threshold" owner ijbrown
Router(config)#rmon alarm 6 ifEntry.14. 3 300 delta rising-threshold  5 7 falling-
threshold  -1 7  owner ijbrown
Router(config)#rmon alarm 7 ifEntry.20. 3 300 delta rising-threshold  5 7 falling-
threshold  -1 7  owner ijbrown
Router(config)#end
Router#
```

If you choose to monitor interface-specific MIBS, such as utilization or errors, then we strongly suggest that you implement Cisco's interface *persist* command, which we discuss in Recipe 17.12.

17.20.3 Discussion

IOS includes some extremely helpful but seldom used remote monitoring (RMON) functionality. The main advantages to configuring the router to monitor its own performance are resource savings and flexibility.

The more conventional alternative is to have a centralized performance server that uses SNMP to periodically poll routers for health statistics. On a large network, this can consume a lot of bandwidth. However, if you move the polling functionality into the router itself, you can get the same benefits without the same bandwidth requirements.

The second major advantage of having the router monitor its own health is flexibility. You can configure each router to monitor itself using the thresholds and parameters that are most meaningful to that router. Since most performance monitoring packages on the market today limit the amount of flexibility in assigning thresholds, this built-in monitoring technique can often give more reliable results.

Of course, the biggest advantage of using RMON is that it is readily available in Cisco's IOS and requires no extra software. If your management software is capable of accepting SNMP traps, no further tools are required. For the budget-conscious administrator, RMON provides detailed performance threshold management without the added cost of performance software.

Cisco's RMON module comes standard in Cisco IOS and is included in the base IP Only feature set. The purpose of RMON is to monitor a certain MIB object on the router and notify the system administrator if the object's value leaves a defined range. It does this by polling itself internally for the values of these MIB objects. You can configure the internal polling interval.

To prevent unnecessary trouble notifications, Cisco RMON alarms use a concept of rising and falling thresholds. An RMON event fires when a value of an SNMP object exceeds the value assigned to the rising threshold. However, subsequent polls that exceed the rising threshold will not trigger an event firing until a polled value drops below the falling threshold value. This concept of rising and falling thresholds prevents the agent from flooding the server with redundant events.

Our examples show how to configure RMON to monitor CPU utilization, low memory situations, buffer failures, link utilization, and interface error counts. This is far from an exhaustive list of the RMON capabilities. Since the router's RMON capabilities rely on the polling of SNMP MIB objects, you can configure it to monitor anything that has an MIB variable. The potential number of statistics you can configure the router to watch is enormous.

The first step in configuring RMON on a router is to define the RMON event(s) that you wish to monitor. In our first RMON example, we configured the router to monitor CPU utilization. The follow two commands tell the router what to do when the CPU load rises above the threshold, and what to do when it falls back into the normal range:

```
Router(config)#rmon event 1 log trap  ORATRAP description "CPU on Router has exceeded
threshold" owner ijbrown
Router(config)#rmon event 2 log description " CPU on Router has normalized" owner
ijbrown
```

The following IOS command defines an RMON event:

```
rmon event number [log] [trap community] [description string] [owner string]
```

This *number* keyword assigns a unique identification number for the RMON event. RMON event numbers range between 1 and 65,535. The optional*log* keyword tells the agent to create a log entry, as well as an SNMP trap, when event is triggered.

A community string accompanies the optional trap keyword. This configures the agent to send an SNMP trap when the event is fired, and assigns the SNMP community string for the trap.

You can specify an optional description for an RMON event using the *description* keyword. The text that follows this keyword is sent with the trap.

This *owner* keyword specifies the owner of an event. In our examples the owner is *ijbrown* . If you do not have AAA usernames configured on your router, you should use the default *admin* user ID.

In this example, RMON Event 1 creates a log entry and an SNMP trap when the CPU utilization on the router exceeds a certain threshold. RMON Event 2 creates a log entry, but doesn't send a trap when the CPU utilization returns to normal.

To view the configured RMON events, use the following command:

```
Router>show rmon events
Event 1 is active, owned by ijbrown
 Description is CPU on Router has exceeded threshold
 Event firing causes log and trap to community ORATRAP, last fired 00:00:00
Event 2 is active, owned by ijbrown
 Description is CPU on Router has normalized
 Event firing causes log, last fired 2w2d
 Current log entries:
      index       time   description
          1       2w2d   CPU on Router has normalized
Router>
```

To make the RMON events meaningful, you also must create a corresponding RMON alarm that defines the conditions for each of the event conditions that we just discussed:

```
Router(config)#rmon alarm 1 lsystem.57.0 60 absolute rising-threshold 70 1 falling-
threshold 402 ownerijbrown
```

The syntax of this command is as follows:

```
rmon alarm number variable interval {absolute | delta}
rising-threshold value [event-number]
falling-threshold value [event-number] [owner string]
```

The *number* in this command is simply a value between 1 and 65,535 that uniquely identifies this alarm.

You can specify a particular MIB object that you want the router to monitor with the *variable* field. In this example, we are polling the Cisco CPU utilization MIB entry, *lsystem.57.0* .

The next value, *interval* , tells the router how often you want it to poll the MIB object. The value can be between 1 and 4,294,967,295 seconds. This example tells the router to poll on a 60-second cycle. Bear in mind that polling too frequently will affect your router's CPU utilization, while polling too slowly

might cause you to miss important events.

Once the data has been polled, you have the option of treating the value as an *absolute* or *delta* . In this example, we are interested in the actual value of the MIB variable, so we specified the *absolute* keyword. The keyword delta tells the router to take the numerical difference between the current sample and the previous one. This feature allows you to determine when a variable's rate of change is too fast or too slow.

You can set a rising threshold value using the *rising-threshold* keyword, which takes an argument between -2,147,483,648 and 2,147,483,647. In this example, the rising threshold is 70%, meaning that the router will trigger an event when the CPU utilization rises through 70%. When this happens, you want the router to trigger a particular event. This association with a particular event is made with the *event-number* argument. In this example, if the rising threshold is exceeded, RMON Event 1 will be triggered.

You configure a falling threshold similarly using the *falling-threshold* keyword. In this example, the falling threshold is 40%, meaning that if the CPU utilization dips below 40%, an event is triggered. As with rising thresholds, you can associate this alarm with a particular event number. In this example, the falling threshold triggers Event 2.

Finally, you can specify an owner for an alarm by using the*owner* keyword.

The example configures RMON Alarm 1 to poll the MIB entry for CPU utilization every minute. If the utilization exceeds 70%, the router will send a trap and create a log message. When the utilization dips back below 40%, it will create a log message.

To view the configured RMON alarms, use the following command:

```
Router>show rmon alarms
Alarm 1 is active, owned by ijbrown
 Monitors lsystem.57.0 every 60 second(s)
 Taking absolute samples, last value was 0
 Rising threshold is 70, assigned to event 1
 Falling threshold is 40, assigned to event 2
 On startup enable rising or falling alarm
Router>
```

Cisco's RMON functionality currently supports SNMP traps, but not SNMP informs.

17.20.4 See Also

Recipe 17.13

Top

Recipe 17.21 Enabling SNMPv3

17.21.1 Problem

You want to enable SNMPv3 on your router for security purposes.

17.21.2 Solution

SNMPv3 supports three modes of operation, each with different security features. These modes were summarized in Table 17-1 at the beginning of this chapter. The following configuration commands enable SNMPv3 with no authentication and no encryption services (noAuthNoPriv):

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#snmp-server view TESTV3 mib-2 include
Router(config)#snmp-server group NOTSAFE v3 noauth read  TESTV3
Router(config)#snmp-server user WEAK NOTSAFE v3
Router(config)#end
Router#
```

Use the following configuration commands to enable SNMPv3 with MD5 authentication and no encryption services (authNoPriv):

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#snmp-server view TESTV3 mib-2 include
Router(config)#snmp-server group ORAROV3 v3 auth read  TESTV3
Router(config)#snmp-server user cking ORAROV3  v3 auth md5 daytona19y
Router(config)#end
Router#
```

And you can enable SNMPv3 with MD5 authentication and DES encryption services (authPriv) as follows:

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#snmp-server view TESTV3 mib-2 include
Router(config)#snmp-server group ORAROV3 v3 auth read  TESTV3
Router(config)#snmp-server user bpugsley ORAROV3  v3 auth md5  hockeyrules priv des56
shortguy
Router(config)#end
Router#
```

17.21.3 Discussion

The IETF has recently approved SNMP Version 3 (SNMPv3) as a full standard and moved SNMPv1 and SNMPv2 to historic status. Essentially, SNMPv3 just acts like a set of security extensions to SNMPv2c, without providing much new core management functionality. All MIB objects and their associated OIDs remain the same from Versions 1 to 3 (with the small exception of the 64-bit counters that were introduced in Version 2). We will focus our attention on the new security features in Version 3.

Security has traditionally been the Achilles heel of SNMP. The security model for Versions 1 and 2c was little more than a simple password sent through the network as clear-text. SNMP required a security facelift to continue to be useful in the future.

SNMPv3 is standards-based network management protocol that is interoperable between vendors. It provides a secure access to devices by providing authentication and encryption of SNMP packets throughout the network. To do this, SNMPv3 needed to include the following security features: authentication, message integrity, and encryption

- Authentication ensures that the messages have originated from a valid source. It proves the authenticity of the packet's source.
- Message integrity ensures that a packet has not been tampered with during transmission.
- Encryption encodes the contents of the packet to prevent unauthorized people from viewing them.

SNMPv3 provides three security levels: *noAuthNoPriv* , *authNoPriv* , and *authPriv* .

- noAuthNoPriv uses a username for authentication and most closely emulates the SNMPv1 and SNMPv2c authentication schemes of transmitting credentials in clear-text. We do not recommend this level of SNMPv3, because it provides no significant advantage over SNMPv2c. If the advanced security features of SNMPv3 are not required for your implementation, it is probably easier to use SNMPv1 or SNMPv2c.
- authNoPriv provides authentication based on the MD5 or SHA algorithms. This security model provides packet authentication and message integrity, but no encryption services. Since SNMP packets are authenticated and cannot be altered in transit, this level of security is sufficient for most organizations.
- authPriv provides the same MD5 or SHA authentication asauthNoPriv . In addition, authPriv allows you to encrypt SNMP packets using 56-bit DES encryption, so packet contents cannot be viewed without authorization. This provides the maximum security available by combining authentication, message integrity, and encryption. The authPriv level of security is suitable for implementations that need to send SNMP packets through the public Internet.

All three SNMPv3 security models require the same three-step process to configure them. First, you must define an SNMP view. Second, you need to create an SNMP group. Third, you create an SNMP user

profile and assign it to a group.

Defining an SNMP view for SNMPv3 is no different that creating a view for SNMPv1 or SNMPv2c. In fact, if there are existing SNMP views on the router that were created for SNMPv1 or SNMPv2c, you can use them with SNMPv3 as well. For more information on creating SNMP views, please see Recipe 17.7 .

For example, here is a simple SNMP view that allows full access to the MIBII tree:

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#snmp-server view TESTV3 mib-2 include
Router(config)#end
Router#
```

To define an SNMPv3 group, use the following command:

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#snmp-server group ORAROV3 v3 auth read TESTV3
Router(config)#end
Router#
```

In this example, we have created a group named *ORAROV3* , which we have configured as an SNMPv3 group (hence the *v3*). We have configured this group to require authentication and assigned it to SNMP view *TESTV3* . Note that we have not assigned a write view to this group, which means that all users assigned to this group will be limited to read-only access. However, the *snmp-server group* command will also allow you to define a read and a write view at the same time. For example:

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#snmp-server view TESTRO mib-2 include
Router(config)#snmp-server view TESTRW system include
Router(config)#snmp-server group TESTGRP v3 auth read TESTRO write TESTRW
Router(config)#end
Router#
```

In this example, we defined two separate SNMP views (*TESTRO* and *TESTRW*) and assigned them to our group. Note, however, that you can assign the same SNMP view to both the read-only access and read-write groups.

To define an SNMPv3 user, use the following command:

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#snmp-server user bpugsley ORAROV3 v3 auth md5 hockeyrules priv des56
shortguy
Router(config)#end
Router#
```

In this example, we have created a user named *bpugsley* and assigned that user to our group named *ORAROV3* . This user will inherit the qualities that we have configured for that group. We have also defined that our user will use the MD5 algorithm for authentication purposes and assigned an authentication password of *hockeyrules* . We have also configured our user to use the optional DES56 packet encryption with the password *shortguy* to provide maximum security. Note that this command, once entered, will not be viewable using the *show running-config* command. We suspect that this is for security purposes.

To view existing SNMP groups, use the *show snmp group* command:

```
Router#show snmp group

groupname: ORAROV3                          security model:v3 auth
readview :TESTV3                            writeview: <no writeview specified>
notifyview: <no notifyview specified>
row status: active

Router#
```

In this example, the group *ORAROV3* is assigned to the security model *v3 auth* , the read-only view is *TESTV3* , and no read-write view exists.

To view the configured SNMPv3 users, use the following command:

```
Router#show snmp user
User name: bpugsley
Engine ID: 800000090300001967OB770
storage-type: nonvolatile        active

Router#
```

Unfortunately, this command provides very little useful information. Apart from confirming if a user exists or not, the output does not display which group the user belongs to, or if the user is configured to use authentication or encryption. When you consider that Cisco's IOS also hides the user SNMP commands from the running configuration, it becomes clear that managing SNMPv3 users is a difficult task. We hope that Cisco will change the output of this command in upcoming releases as SNMPv3 becomes more popular.

17.21.4 Using the SNMPv3 Security Levels

You can extract SNMP information from the router using each of the three SNMPv3 security levels. We will use NET-SNMP's *snmpget* command, which has full SNMPv3 support.

In our first example (noAuthNoPriv), we will poll the router for its system name using a standard MIB-II object, *sysName* .

```
Freebsd% snmpget -v3 -u WEAK -l noAuthNoPriv  Router  sysName.0
```

```
system.sysName.0 = Router.oreilly.com
Freebsd%
```

Note that no user password was supplied, so the router simply accepted the user ID *WEAK* for authentication purposes. This user ID was sent through the network in clear-text. This command has also introduced two new attributes for the *snmpget* command: *-u* and *-l* . The *-u* attribute allows you to specify the security name, while *-l* defines the security level.

The next example uses the *authNoPriv* security model. We will poll the exact same MIB object using MD5 authentication:

```
Freebsd% snmpget -v3 -u cking -l authNoPriv -a MD5 -A daytona19y Router sysName.0
system.sysName.0 = Router.oreilly.com
Freebsd%
```

In this example, we specify a user password *daytona19y* using the *-A* option and an the authentication protocol *MD5* with the *-a* option. SNMPv3 uses the authentication protocol to authenticate users without sending the password in clear-text. It is important to notice that the result of this SNMP Get is the same as our first example. However, we gathered the information in a much more secure manner. In fact, the same MIB object (*sysName*) could be retrieved using SNMPv1 if the router was configured to accept the request. However, this method would be considerably less secure.

The final example illustrates how to poll an MIB object using the authentication and encryption services of the *authPriv* security model:

```
Freebsd% snmpget -v3 -u bpugsley -l authPriv -a MD5 -A hockeyrules -x DES -X shortguy
Router sysName.0
system.sysName.0 = Router.oreilly.com
Freebsd%
```

In this example, we added two new variables, privacy protocol type DES using *-x DES* and a privacy protocol pass phrase with *-X shortguy* . These variables enable SNMPv3 packet encryption and specify the pass phrase to use. This ensures that prying eyes cannot view the packet contents in transit. To illustrate the effectiveness of SNMPv3's encryption service, we provide a captured SNMPv3 packet. The packet was captured using the Ethereal protocol analyzer (for more information on Ethereal please see Appendix A):

```
Simple Network Management Protocol
    Version: 3
    Message Global Header
        Message Global Header Length: 16
        Message ID: 1608369049
        Message Max Size: 1480
        Flags: 0x03
            .... .0.. = Reportable: Not set
            .... ..1. = Encrypted: Set
            .... ...1 = Authenticated: Set
        Message Security Model: USM
```

```
Message Security Parameters
    Message Security Parameters Length: 58
     Authoritative Engine ID: 800000090300000019670B780
    Engine Boots: 2
    Engine Time: 1469970
    User Name: bpugsley
     Authentication Parameter: B53EFA21230735541B207A39
    Privacy Parameter: 00000002C483B016
Encrypted PDU (74 bytes)
```

The packet response from the router contains some useful SNMP information, such as current version, encryption enabled, authentication enabled, and username (*bpugsley*), but Ethereal is unable to decipher the payload (*Encrypted PDU*). This is significant because the other versions of SNMP—including the other security models within SNMPv3—transport payload information in clear-text. At last, SNMP has evolved into a secure protocol.

Of course, SNMPv3 also provides full support for traps and informs including authentication, message integrity, and encryption. SNMPv3 traps and informs support the same three models of security as inbound services do. However, the *noAuthNoPriv* model provides no tangible advantage over SNMPv1 or SNMPv2c, and the *authPriv* model tends to be overkill since few networks will require encrypted traps.

To enable SNMPv3 trap support using authentication and message integrity, use the following command:

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#snmp-server host 172.25.1.1 version 3 auth ijbrown snmp envmon
Router(config)#end
Router#
```

The process of enabling SNMPv3 traps or informs is similar to the process for SNMPv2c, but with a few minor twists. First, you must define a SNMPv3 group and user, as in the previous examples. Second, you must include the keyword *auth* , which enables authentication. Third, you must include a valid SNMPv3 user (*ijbrown* , in this case). The router is then capable of forwarding SNMPv3 traps with full SNMPv3 authentication and message integrity enabled. For more information on enabling SNMP traps in general, see Recipe 17.13 .

17.21.5 See Also

Recipe 17.1 ; Recipe 17.13 ; Recipe 17.7

Top

Recipe 17.22 Using SAA

17.22.1 Problem

You want to configure the routers to automatically poll one another to collect performance statistics.

17.22.2 Solution

Cisco supplies a feature called the Service Assurance Agent (SAA) in IOS Version 12.0(5)T and higher, which allows the routers to automatically poll one another to collect end-to-end performance statistics:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#rtr responder
Router1(config)#rtr 10
Router1(config-rtr)#type echo protocol ipIcmpEcho 10.1.2.3
Router1(config-rtr)#tag ECHO_TEST
Router1(config-rtr)#threshold 1000
Router1(config-rtr)#frequency 300
Router1(config-rtr)#exit
Router1(config)#rtr schedule 10 life 2147483647 start-time now
Router1(config)#rtr 20
Router1(config-rtr)#type jitter dest-ipaddr 10.1.2.3 dest-port 99 num-packets 100
Router1(config-rtr)#tag JITTER_TEST
Router1(config-rtr)#frequency 300
Router1(config-rtr)#exit
Router1(config)#rtr schedule 20 life 100000 start-time now ageout 3600
Router1(config)#exit
Router1#
```

The target router (`10.1.2.3`), which is specified as the destination in both of these tests, must be configured to respond to SAA tests:

```
Router2#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router2(config)#rtr responder
Router2(config)#exit
Router2#
```

17.22.3 Discussion

The SAA feature includes replace the earlier Round Trip Reporter (RTR) and Route Trip Time Monitor

(RTTMON) facilities, which were available in IOS Version 11.3, and use the same basic syntax. However, where RTR includes only some simple round-trip ping and SNA tests, SAA includes several more interesting and useful features.

The first line in the example is the *rtr responder* command. This is required on all routers that will be taking part in SAA, including the targets of these tests. Both of these examples use a target IP address of `10.1.2.3`. This destination must be another Cisco router that is also configured with the *rtr responder* command.

In this example, we have configured two tests. The first test is given the arbitrary number 10 and the name *ECHO_TEST*. The second test is number 20, and is called*JITTER_TEST*. Note that you don't actually need to give your SAA tests names, but it is a good idea if you have several of them. This name (or tag) is included in the SAA SNMP MIB table for this test. So, if you intend to download the test data via SNMP for performance management purposes, it can be extremely useful to name your tests so you know what they are.

Let's look at both of these example tests in more detail.

The first test does an ICMP echo (ping) to the destination device, `10.1.2.3`:

```
Router1(config)#rtr 10
Router1(config-rtr)#type echo protocol ipIcmpEcho 10.1.2.3
Router1(config-rtr)#tag ECHO_TEST
Router1(config-rtr)#threshold 1000
Router1(config-rtr)#frequency 300
Router1(config-rtr)#exit
```

The *threshold* command defines a minimum interesting threshold, which is set to 1,000 milliseconds here. This allows you to count the number of *ping* tests in which the round trip time was greater than one second, in addition to keeping track of the *ping* times and number of *ping* failures, which we will show in a moment.

Next is the *frequency* command, which defines how often this test will be run in seconds. In this case, we want the test to run every 5 minutes (300 seconds).

Then, once you have defined the test in the *rtr* configuration block, you have to tell the router when to run it. This is done with the *rtr schedule* command:

```
Router1(config)#rtr schedule 10 life 2147483647 start-time now
```

This command defines the schedule for running test number 10. It sets a lifetime for this test of 2,147,483,647 seconds (a very long time), which is the maximum value. This effectively means that this test will continue to run indefinitely. It is scheduled to start immediately.

When we scheduled the second test we used slightly different parameters:

```
Router1(config)#rtr schedule 20 life 100000 start-time now ageout 3600
```

In this case, the test is scheduled to run only for 100,000 seconds, which is about 27 hours. We have also configured an *ageout* value of 3600 seconds for this test. This says that the router will keep this test rule in memory for this length of time after it expires. The *ageout* option allows you to restart the test without having to reconfigure it.

You can view the data for the first test as follows:

```
Router1#show rtr operational-state 10
        Current Operational State
Entry Number: 10
Modification Time: 18:51:53.000 EST Tue Dec 17 2002
Diagnostics Text:
Last Time this Entry was Reset: Never
Number of Octets in use by this Entry: 1910
Connection Loss Occurred: FALSE
Timeout Occurred: FALSE
Over Thresholds Occurred: FALSE
Number of Operations Attempted: 203
Current Seconds Left in Life: 2147483647
Operational State of Entry: active
Latest Completion Time (milliseconds): 54
Latest Operation Start Time: 11:41:53.000 EST Wed Dec 18 2002
Latest Operation Return Code: ok
Latest 10.1.2.3
```

In this output you can see that it has run this test 203 times, and that the last test took 54 milliseconds and completed successfully. Note that it doesn't give a running average *ping* time. However, one of the nicest features of SAA is that you can configure a network management station to download this data using SNMP, provided you have the SAA MIB loaded on your server.

The second test is considerably more interesting. This test measures jitter between the routers by sending a series of UDP packets and looking at the time differences between them at both ends:

```
Router1(config)#rtr 20
Router1(config-rtr)#type jitter dest-ipaddr 10.1.2.3 dest-port 99 num-packets 100
Router1(config-rtr)#tag JITTER_TEST
Router1(config-rtr)#frequency 300
Router1(config-rtr)#exit
Router1(config)#rtr schedule 20 life 100000 start-time now ageout 3600
```

The *type* command defines a jitter test to the same destination IP address as the previous test. In this case, we have decided to use UDP port 99 for our test, and each test run will consist of 100 packets. The *frequency* command tells the router to run this test every five minutes. Here is some sample output from this test:

```
Router1#show rtr operational-state 20
        Current Operational State
Entry Number: 20
```

```
Modification Time: 10:25:36.000 EST Wed Dec 18 2002
Diagnostics Text:
Last Time this Entry was Reset: Never
Number of Octets in use by this Entry: 1742
Number of Operations Attempted: 22
Current Seconds Left in Life: 93400
Operational State of Entry: active
Latest Operation Start Time: 12:10:36.000 EST Wed Dec 18 2002
RTT Values:
NumOfRTT: 98    RTTSum: 6063    RTTSum2: 384317
Packet Loss Values:
PacketLossSD: 0 PacketLossDS: 2
PacketOutOfSequence: 2  PacketMIA: 0    PacketLateArrival: 0
InternalError: 0        Busies: 0
Jitter Values:
MinOfPositivesSD: 4     MaxOfPositivesSD: 14
NumOfPositivesSD: 32    SumOfPositivesSD: 175   Sum2PositivesSD: 1111
MinOfNegativesSD: 1     MaxOfNegativesSD: 5
NumOfNegativesSD: 60    SumOfNegativesSD: 175   Sum2NegativesSD: 547
MinOfPositivesDS: 1     MaxOfPositivesDS: 45
NumOfPositivesDS: 20    SumOfPositivesDS: 78    Sum2PositivesDS: 2166
MinOfNegativesDS: 1     MaxOfNegativesDS: 16
NumOfNegativesDS: 21    SumOfNegativesDS: 69    Sum2NegativesDS: 693
```

There is a clearly a lot more information in this test output. This is because measuring jitter is not a simple single-variable test. A jitter measurement characterizes the statistical distribution of packet-by-packet variation in forward and backward latency, as well as for the round trip. Note that, as with the SAA *ping* test we discussed earlier, the router records only the results of the most recent test. If you want to keep historical records, you need to poll and download the SAA MIB tables once per poll cycle.

The first set of numbers are the Round Trip Time (RTT) values. You can see that this sample included 98 packets. The total of all of the round trip times of all of these packets was 6063 milliseconds, and the sum of the squares of all of these times was 384,317 milliseconds. These values are not extremely meaningful in themselves, but if you divide the RTTSum value by the number of measurements, you get the average latency for this set of packets: roughly 61 milliseconds.

Applying some simple statistics, you can use the square value to understand how the actual values are spread around this average. The mean of the squares of the round trip times is 3,922 milliseconds^2 (just dividing the sum of the squares by the total number of samples). If you subtract the square of the average from this value and take the square root, you get a statistical estimate of the variation in milliseconds. The higher this value, the greater the spread. In this case, you can calculate that this spread is roughly 14 milliseconds. This means that half of the time, the round trip latency is within the range 61±14ms. Note that the ± symbol is a standard mathematical notation that, in this case, indicates a range from 47ms (61-14) to 75ms (61+14).

The next set of data records dropped packets. Recall that the sample size is 100 packets, but the

NumOfRTT value is only 98. So the network must have dropped two of our test packets. SAA separately keeps track of packets lost in both directions, source to destination (PacketLossSD) and destination to source (PacketLossDS). This router is the source; the other router is the destination. So, in this example, both of the lost packets happened on the way back. Also note that the output claims that there were two out-of-sequence packets during this test, which is consistent with the number of dropped packets. The router saw the sequence number jump by 2 because of the dropped packet, and recorded it as an out-of-sequence error.

The next group of numbers includes the actual jitter measurements. There are two groups of numbers here. The variables that end with "SD" are measured from the source to the destination, and those labeled "DS" are for the return path. Within each of these groups there are two subgroups, one for "Positives," and the other for "Negatives." "Positives" are events where the spacing between two packets has increased since the last pair of successive packets. The "Negatives" counters record all of the times that the interpacket spacing decreased. Let's look a little bit more closely at one set of values:

```
MinOfPositivesSD: 4      MaxOfPositivesSD: 14
NumOfPositivesSD: 32     SumOfPositivesSD: 175    Sum2PositivesSD: 1111
```

Of the 100 packets the router sent in this polling interval, there were 32 cases in which the jitter in the forward direction had a positive value. Of these, the largest value was 14 milliseconds and the smallest was 4 milliseconds. We can use the sum and the sum of the squares to calculate the average and spread of values in precisely the same way as we did to calculate the average latency. The result we get is that half of the time, positive jitter in this direction was within the range 5.5±2.2ms.

Applying this same technique to the other jitter measurements gives other useful statistics. The negative jitter from source to destination was 2.9±0.8ms with a maximum of 5ms and a minimum value of 1ms. In the other direction, the positive jitter was 3.9±9.6ms and the negative jitter was 3.3±4.7ms. These last two values might look a little bit funny because the spread is larger than the mean. However, this is not actually bad because the output also shows that the maximum positive jitter in this direction was 45ms, and the maximum negative jitter was 16ms. The spread is very large, but the mean jitter values are relatively small. This is a fairly typical result.

Top

Chapter 18. Logging

Top

Introduction

Many network administrators overlook the importance of router logs. Logging is critical for fault notification, network forensics, and security auditing.

Cisco routers handle log messages in five ways:

- By default, the router sends all log messages to its console port. Only users that are physically connected to the router console port may view these messages. This is called console logging.
- Terminal logging is similar to console logging, but it displays log messages to the router's VTY lines. This type of logging is not enabled by default; if you want to use it, you need to need activate it for each required line.
- Buffered logging creates a circular buffer within the router's RAM for storing log messages. This circular buffer has a fixed size to ensure that the log will not deplete valuable system memory. The router saves memory by deleting old messages from the buffer as new messages are added.
- The router can use syslog to forward log messages to external syslog servers for centralized storage. This type of logging is not enabled by default. Much of this chapter is devoted to configuring remote syslog features. The router sends syslog messages to the server on UDP port 514. The server does not acknowledge these messages.
- With SNMP trap logging, the router is able to use SNMP traps to send log messages to an external SNMP server. This is an effective method of handling log messages in a SNMP-based environment, but it has certain limitations. We discuss this logging method in Chapter 17, which deals with SNMP configuration.

Cisco log messages are categorized by severity level, following the structure and format of the BSD Unix syslog framework, as shown in Table 18-1. The lower the severity level, the more critical the log message is.

Table 18-1. Cisco logging severity levels

Level	Level name	Description	Syslog definition
0	Emergencies	Router unusable	LOG_EMERG

Level	Level name	Description	Syslog definition
1	Alerts	Immediate action needed	LOG_ALERT
2	Critical	Critical conditions	LOG_CRIT
3	Errors	Error conditions	LOG_ERR
4	Warnings	Warning conditions	LOG_WARNING
5	Notifications	Normal but important conditions	LOG_NOTICE
6	Informational	Informational messages	LOG_INFO
7	Debugging	Debugging messages	LOG_DEBUG

Here is an example of a log message that shows the typical format of Cisco router log messages:

```
Apr 12 14:01:16: %CLEAR-5-COUNTERS: Clear counter on all interfaces by ijbrown on
vty0 (172.25.1.1)
```

The log message is broken into three sections that are delimited by colons. The first section is the optional date and time section that is enabled by using the *service timestamp* configuration command. A detailed discussion of timestamps can be found in Chapter 14.

The second part of the log message, *%CLEAR-5-COUNTERS*, gives the message's code and severity level. The message code family is *CLEAR*, and the priority level is *-5-* (indicating a *Notifications* severity level message). The family type is *COUNTERS*. All Cisco log messages are arranged in this manner. There are many different message codes, such as *FRAME* for Frame Relay messages, *SYS* for system messages, and *LINK* for interface messages. Within each message code, log messages are categorized by severity type. 7 is the least severe, while 0 is the most critical (following the syslog model). Finally, each specific message type is assigned a unique message code such as *COUNTERS* in this case, or *UPDOWN* for *LINK* messages, and so forth.

The remainder of a log entry is the message body, which contains human-readable text. The example message above contains the message body "Clear counter on all interfaces by *ijbrown* on *vty0* (`172.25.1.1`)." The message body generally contains easy-to-understand text, as well as some custom variables (in this case, *ijbrown* and *vty0*), which helps to make log messages more meaningful.

Table 18-2 shows an example of a typical log message for each of the eight severity levels.

Table 18-2. Sample router log messages

Level	Level name	Sample router messages
0	Emergencies	System shutting down due to missing fan tray
1	Alerts	Core CRITICAL temperature limit exceeded
2	Critical	Memory allocation failures
3	Errors	Interface up/down messages
4	Warnings	Configuration file written to server, via SNMP request
5	Notifications	Line protocol up/down
6	Informational	Access-list violation logging
7	Debugging	Debug messages

You will rarely see log messages with severity levels of Alert or Emergency—any problems this severe generally mean that the router is inoperable.

Top

Recipe 18.1 Enabling Local Router Logging

18.1.1 Problem

You want your router to record log messages, instead of just displaying them on the console.

18.1.2 Solution

Use the *logging buffered* configuration command to enable the local storage of router log messages:

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#logging buffered informational
Router(config)#end
Router#
```

18.1.3 Discussion

This feature causes the router to store all log messages to a revolving buffer called the logging buffer. Many network administrators find it convenient and useful to keep detailed router logs on the router itself. The router discards its oldest messages to make room for new ones. This ensures that the logging buffer contains the most recent messages without depleting the router's RAM. You can use the *show logging* command to view this buffer:

```
Router>show logging
Syslog logging: enabled (0 messages dropped, 0 messages rate-limited, 0 flushes, 0
overruns)
    Console logging: level debugging, 653 messages logged
    Monitor logging: level debugging, 65 messages logged
    Buffer logging: level informational, 1 messages logged
    Logging Exception size (4096 bytes)
    Trap logging: level informational, 657 message lines logged
Log Buffer (4096 bytes):
Mar 26 09:02:25: %SEC-6-IPACCESSLOGS: list 99 denied 172.16.2.2 5 packets
Mar 26 09:04:56: %CLEAR-5-COUNTERS: Clear counter on all interfaces on vty1
Mar 26 09:05:13: %SYS-5-CONFIG_I: Configured from console by ijbrown on vty1
Router>
```

Note that the default severity logging level is set to debugging. You can adjust the severity level of the buffered log with the *severity level* keyword. In the solution section example, we configured the router with the keyword *informational* . This will cause it to ignore debugging messages, but retain all other system log messages.

The log messages appear in order from oldest to most recent. By default, the *show logging* command displays all messages contained in the log buffer. However, you can display specific message types can by using *output modifiers* :

```
Router>show log | include denied
Apr  7 21:16:12 EDT: %SEC-6-IPACCESSLOGS: list 98 denied 172.25.25.1 19 packets
Apr  7 21:21:12 EDT: %SEC-6-IPACCESSLOGS: list 98 denied 172.25.1.5 1 packet
Apr  7 21:26:12 EDT: %SEC-6-IPACCESSLOGS: list 98 denied 172.25.25.1 19 packets
Apr  7 21:31:13 EDT: %SEC-6-IPACCESSLOGS: list 98 denied 172.25.25.1 5 packets
Apr  7 21:33:13 EDT: %SEC-6-IPACCESSLOGS: list 98 denied 172.25.1.5 16 packets
Apr  7 21:36:13 EDT: %SEC-6-IPACCESSLOGS: list 98 denied 172.25.25.1 5 packets
Router>
```

By using output modifiers, you can display a single type of message based on a *regular expression* , similar to the *grep* command in Unix.

We discussed the importance of accurate timekeeping and log timestamping in Chapter 14 , where we recommended enabling log timestamps to help make the log messages more meaningful.

To disable the router's logging buffer, use the following command:

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#no logging buffered
Router(config)#end
Router#
```

18.1.4 See Also

Recipe 18.2 ; Recipe 18.3 ; Chapter 14

Top

Recipe 18.2 Setting the Log Size

18.2.1 Problem

You want to change the size of the router's log.

18.2.2 Solution

You can use the optional size attribute with the *logging buffered* configuration command to change the size of your router's internal log buffer:

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#logging buffered 16000
Router(config)#end
Router#
```

Adjusting the size of the router's logging buffer wipes out all of the current contents of the buffer.

18.2.3 Discussion

The typical default size of a router's logging buffer is 4096 bytes (although some high-end routers will default to a higher value). A buffer of this size can hold approximately 50 log messages before overwriting occurs. 50 messages, although better than no logging, is relatively small and most engineers will want increase their buffer size to store more messages. To check the size of your router's logging buffer, use the *show buffer* command:

```
Router>show logging
Syslog logging: enabled (0 messages dropped, 0 messages rate-limited, 0 flushes, 0
overruns)
    Console logging: level debugging, 653 messages logged
    Monitor logging: level debugging, 65 messages logged
   Buffer logging: level debugging, 1 messages logged
    Logging Exception size (4096 bytes)
    Trap logging: level informational, 657 message lines logged

Log Buffer (16000 bytes):
Router>
```

As you can see, this router's buffer size is currently set to 16,000 bytes (roughly 16KB).

The router will theoretically accept a wide range of buffer sizes ranging from 4,096 bytes (nothing smaller) to an astronomical 2,147,483,647 bytes (about 2GB). Exercise caution when choosing the size

of your logging buffer because it comes out of the router's system memory. A good rule is to set your logging buffer to 16KB for smaller routers. Routers with more than 32MB of memory can safely dedicate 32KB, or even 64KB without problems. To be safe, always check the amount of free memory on your router with the *show memory* command before increasing your buffer size.

You can also combine the keywords in Recipe 18.1and Recipe 18.2 into a single configuration command:

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#logging buffered 16000 informational
Router(config)#end
Router#
```

In this case, we set the buffer size to 16,000 bytes and the severity level to informational with a single configuration command.

18.2.4 See Also

Recipe 18.1 ; Recipe 18.2

Top

Recipe 18.3 Clearing the Router's Log

18.3.1 Problem

You want to clear the router's log buffer.

18.3.2 Solution

Use the *clear logging* command to clear the router's internal log buffer:

```
Router#clear logging
Clear logging buffer [confirm]<enter>
Router#
```

18.3.3 Discussion

There are times when it is useful to be able to clear your router's internal buffer log. For instance, while debugging a network problem, old debug messages can cause unnecessary confusion during stressful moments. This command will clear the router log:

```
Router#clear logging
Clear logging buffer [confirm]<enter>
Router>show logging
Syslog logging: enabled (0 messages dropped, 0 messages rate-limited, 0 flushes, 0
overruns)
    Console logging: level debugging, 64201 messages logged
    Monitor logging: level debugging, 64820 messages logged
    Buffer logging: level debugging, 9 messages logged
    Logging Exception size (4096 bytes)
    Trap logging: level debugging, 2253 message lines logged
       Logging to 172.25.1.1, 2253 message lines logged

Log Buffer (4096 bytes):
Router>
```

This command has no effect on the other forms of logging.

Top

Recipe 18.4 Sending Log Messages to Your Screen

18.4.1 Problem

You want the router to display log messages to your VTY session in real time.

18.4.2 Solution

Use the *terminal monitor* command to enable the displaying of log messages to your VTY:

```
Router#terminal monitor
Router#
```

To disable logging to your VTY session, use the following command:

```
Router#terminal no monitor
Router#
```

18.4.3 Discussion

By default, routers forward all logging messages to their console ports, but not to their VTY lines. When you are troubleshooting a network problem on a remote router, it is often quite useful to instruct the router to send log messages to your VTY so that you can view them in real time. Here is an example showing how to configure the router to display messages with informational severity level and greater (see Table 18-1Table 18-1 for more information about logging severity levels) to VTY lines:

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#logging monitor informational
Router(config)#exit
Router#terminal monitor
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#interface Fastethernet0/0
Router(config-subif)#shutdown
Router(config-subif)#end
Router#
Mar 26 09:36:43: %LINK-5-CHANGED: Interface Fastethernet0/0, changed state to
administratively down
```

This example changes the logging monitor level to *informational* , enables terminal monitoring, and changes the state of an interface to trigger a sample log message. By default, when you enable VTY displaying of log messages, the terminal monitor severity level is set to *debugging* so that you can see

all messages. In this example, we have changed the severity level to *informational* , which will suppress the printing of debug messages but still display messages with all other severity levels. Keep in mind that setting the severity level for VTY logging is a configuration command, while the command to enable VTY logging, *terminal monitor* , is a privileged level EXEC command that must be set per session.

Enabling this type of logging makes a Telnet session to a router's VTY port similar to the process of connecting directly to the router's console port. You can use the *show logging* command to view the current monitor logging configuration:

```
Router>show logging
Syslog logging: enabled (0 messages dropped, 101 messages rate-limited, 0 flushes, 0
overruns)
     Console logging: level debugging, 66712 messages logged
    Monitor logging: level informational, 65263 messages logged
        Logging to: vty66(0)
     Buffer logging: level debugging, 644 messages logged
     Logging Exception size (4096 bytes)
     Trap logging: level debugging, 3805 message lines logged
        Logging to 172.25.1.1, 3805 message lines logged

Log Buffer (8000 bytes):
```

The bold section of the output shows that the monitor logging facility has been set to a severity level of *informational* and that one session is currently in use, with the messages being displayed on vty66(0).

```
Router>show users
    Line       User       Host(s)              Idle       Location
* 66 vty 0     ijbrown    idle                 00:00:00 freebsd.oreilly.com
  67 vty 1     kdooley    idle                 00:00:26 solaris.oreilly.com
Router>
```

You can easily determine which user is currently using the monitor logging facility by issuing theshow users command. This output indicates that the user *ijbrown* is currently using the monitor logging facility.

> Use caution when enabling VTY logging in conjunction with debugging, as it can overwhelm your session.

This feature is so useful that enabling it will soon become second nature.

Top

Recipe 18.5 Using a Remote Log Server

18.5.1 Problem

You want to send log messages to a remote syslog server.

18.5.2 Solution

Use the following command to send router log messages to a remote syslog server:

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#logging 172.25.1.1
Router(config)#end
Router#
```

Although configuring the router with a static IP address is the preferred method of configuring a syslog server, you can also specify a hostname to be resolved:

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#ip host nms.oreilly.com 172.25.1.1
Router(config)#logging nms.oreilly.com
Router(config)#end
Router#
```

With this configuration, the router will attempt to resolve the server name that is provided. If the router cannot resolve the server name via DNS or static host lookup then the entry will fail. For more information about DNS and static hostnames, see Chapter 2.

18.5.3 Discussion

Forwarding log messages to a remote syslog server has several advantages over just retaining log messages locally on the router. The primary advantage is that messages sent to the server are stored to disk. All other forms of router logging are lost when the router reloads, including vital log messages that occur just before a router crashes due to error.

Another advantage of using a remote syslog server is storage capacity. A router stores logging messages in internal system memory, which severely limits the number of log messages that can be stored. A syslog server, on the other hand, can store days', weeks', or even months' worth of log messages. It is not uncommon for an organization to retain a month or more of archived log messages for later examination.

Finally, being able to view log messages from all of your routers in a single location can be quite useful. Forwarding all router log messages to a common log file can assist in fault isolation, problem resolution, network forensics, and security investigations. In addition, parsing router log files using custom scripts can provide an excellent understanding of network health. Many network management software vendors now include tools to handle syslog messages.

The example below illustrates a router configured with two remote syslog servers:

```
Router>show logging
Syslog logging: enabled (0 messages dropped, 0 messages rate-limited, 0 flushes, 0
overruns)
    Console logging: level debugging, 654 messages logged
    Monitor logging: level debugging, 65 messages logged
    Buffer logging: level debugging, 2 messages logged
    Logging Exception size (4096 bytes)
    Trap logging: level informational, 658 message lines logged
        Logging to 172.25.1.1, 1 message lines logged
        Logging to 172.25.1.3, 1 message lines logged

Log Buffer (4096 bytes):
Router>
```

The syslog protocol uses UDP port 514, and messages are forwarded asynchronously without acknowledgement from the server. In other words, communications between the router and server flow in a single direction with the server acting as a passive receiver.

18.5.4 See Also

Recipe 18.6 ; Recipe 18.9 ; Recipe 18.14

Top

Recipe 18.6 Enabling Syslog on a Unix Server

18.6.1 Problem

You want to configure a Unix server to accept syslog messages from routers.

18.6.2 Solution

For most flavors of Unix and Linux, you simple need to modify the */etc/syslog.conf* file on your Unix server to include the following entry (basic configuration):

```
local7.info                                    /var/log/rtrlog
```

Cisco routers use the *local7* logging facility by default. This configuration line tells the syslog program to store any such messages that have a severity level of *informational* or higher in the file */var/log/rtrlog*. The lefthand column in the configuration file specifies the logging facility and priority level, while the right hand column specifies the logging facility and priority level. The righthand column specifies the file name where these messages should be stored.

> Note that the *syslog.conf* file needs tabs, and not spaces, between the various fields.

18.6.3 Discussion

By default, your syslog server may not be equipped to handle router log messages. The configuration entry show in the example causes the syslog daemon to store all router messages with an *informational* severity level or higher in a file called */var/log/rtrlog*. This file must exist and have the proper file attributes before the server can begin to forward messages to it:

```
Freebsd# cd /var/log
/var/log
Freebsd# touch rtrlog
Freebsd# chmod 644 rtrlog
Freebsd#
```

After changing the *syslog.conf* file, you must reload or *HUP* the syslog daemon to force it to read your new configuration file and begin storing router log messages. On System V-based Unix servers, use the following commands:

```
Solaris# ps -ef | grep syslogd
    root   142     1  0   Nov 12 ?        1:21 /usr/sbin/syslogd -m 30
Solaris# kill -HUP 142
Solaris#
```

On BSD-based Unix and Linux servers, use the following commands:

```
Freebsd# ps -aux | grep syslogd
root       66  0.0  0.2   960  624  ??  Ss    3Mar02   0:28.66 syslogd -m 30
Freebsd# kill -HUP 66
Freebsd#
```

For more information on your syslog daemon and its configuration options, check your system's manpages using the Unix commands *man syslog* and *man syslog.conf*.

Some Unix flavors, including most Linux distributions, require the syslog daemon be initialized with the -r switch before they will accept remote syslog messages. For more information, use the command *man syslogd.*

18.6.4 See Also

Recipe 18.7; Recipe 18.11; Recipe 18.12

Top

Recipe 18.7 Changing the Default Log Facility

18.7.1 Problem

You want to change the default logging facility.

18.7.2 Solution

Use the *logging facility* configuration command to change the syslog facility that the router sends error messages to:

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#logging 172.25.1.1
Router(config)#logging facility local6
Router(config)#end
Router#
```

The default syslog facility setting is *local7*.

18.7.3 Discussion

By default, the router will forward all syslog messages to the server's *local7* log facility. You can modify this behavior and forward all of your router's syslog messages to another facility by utilizing the *logging facility* configuration command. Table 18-1Table 18-1 lists the possible logging facilities that a router will accept.

Table 18-3. Cisco logging facility types

Facility	Description
Auth	Authorization system
Cron	Cron/at facility
Daemon	System daemons
Kern	Kernel

Facility	Description
local0	Local use
local1	Local use
local2	Local use
local3	Local use
local4	Local use
local5	Local use
local6	Local use
local7	Local use (default facility for Cisco routers)
Lpr	Line printer system
Mail	Mail system
News	USENET news
sys9	System use
sys10	System use
sys11	System use
sys12	System use
sys13	System use
sys14	System use
Syslog	Syslog itself
User	User process
Uucp	Unix-to-Unix copy system

> We generally recommend that you choose one of the "local" facilities, as these are intended specifically for this type of use.

There are a number of situations where it can be quite useful to choose a facility other than the default. First, another application on the syslog server itself may already be using the *local7* logging facility. Although most applications provide a way to change the default logging facility some, regrettably, do not. If two application use the same logging facility, the server will merge

Second, you might want to separate log messages from routers and switches or other types of network equipment. This makes parsing through the log files much easier. For example, you could configure your switches to forward all log messages to *local7*, and have your routers use *local6*.

Third, separating perimeter router logs from those of internal company routers can often be important for security auditing reasons. Perimeter routers protect the organization from outsiders and require more diligent attention. Sending their log messages to a separate file so that they are not lumped in with the rest of the organization's router messages makes it easier to give them this extra attention. For instance, perimeter router logs may require different archive periods or have specialized scripts that parse through them. Assigning a different log facility to them is generally a good idea.

The next example shows a sample portion of a *syslog.conf* file that forwards log messages from all perimeter routers to *local5*, all other router logs to *local6*, and all switch logs to *local7*:

```
local5.info                             /var/log/seclog
local6.info                             /var/log/rtrlog
local7.info                             /var/log/switchlog
```

The sample router configuration in the solution section forwards router log messages to log facility *local6*. You can configure the perimeter routers to forward their log messages to log facility *local5* as follows:

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#logging 172.25.1.1
Router(config)#logging facility local5
Router(config)#end
Router#
```

One final useful thing to do with your syslog configuration is to send high-severity log messages to a separate file to make parsing easier. The following example shows a sample *syslog.conf* configuration that logs all router log messages to a single file called */var/log/rtrlog*, and all high severity log messages to a file called */var/log/rtrpriority*:

```
local7.info                             /var/log/rtrlog
local7.err                              /var/log/rtrpriority
```

18.7.4 See Also

Recipe 18.8

Top

Recipe 18.8 Restricting What Log Messages Are Sent to the Server

18.8.1 Problem

You want to limit which logging levels the router will send to the syslog server.

18.8.2 Solution

Use the *logging trap* configuration command to limit the severity level of syslog messages:

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#logging 172.25.1.1
Router(config)#logging trap notifications
Router(config)#end
Router#
```

18.8.3 Discussion

By default, when you enable remote logging on a router, it will forward only those messages with a severity level of *informational* or higher (see Table 18-1). This means that the router forwards everything but debugging messages to the syslog server. Raising the severity of the log messages forwarded to the syslog server can help reduce bandwidth utilization across the network and minimize the disk space required for storing the log messages on the server. This example shows the output of the *show logging* EXEC command:

```
Router>show logging
Syslog logging: enabled (0 messages dropped, 0 messages rate-limited, 0 flushes, 0
overruns)
    Console logging: level debugging, 658 messages logged
    Monitor logging: level debugging, 65 messages logged
    Buffer logging: level debugging, 6 messages logged
    Logging Exception size (4096 bytes)
    Trap logging: level notifications, 662 message lines logged
        Logging to 172.25.1.1, 5 message lines logged
        Logging to 172.25.1.3, 5 message lines logged

Log Buffer (4096 bytes):
Router>
```

Note that the logging severity level is set to *notifications* , and that both outbound servers are limited to the same level. The *logging trap* command sets the severity level for all syslog servers.

Another important use for the logging trap command is to allow the router to send debug level

messages to a syslog server. The default severity level for syslog is *informational* , so the router will not forward any debug messages. Having the ability to store debug messages can often be useful in fault isolation and trouble resolution. The following commands show how to enable the forwarding of *debugging* severity level messages:

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#logging 172.25.1.1
Router(config)#logging trap debugging
Router(config)#end
Router#
```

You must also configure the syslog server to handle debug level messages. You can accomplish this in one of two ways—by creating a dedicated file to handle debug messages, or by forwarding debug messages to the normal router log file. The following two examples will demonstrate how to modify your *syslog.conf* file to handle both situations:

```
local7.debug                    /var/log/debug
```

or:

```
local7.=debug                   /var/log/debug
```

The difference between these two methods is that while the first sends all messages with a severity greater than or equal to debug severity to the indicated file, the second directs only debug level messages to this file. The first example is a more traditional method used to assign debug level messages to a particular file, but the second example is preferred because it allows you to separate debug messages from the normal router log messages. Most modern syslog facilities will handle the second example command syntax, but check your manpages with the *man syslog.conf* command to ensure compatibility.

To simply forward all debug level messages to the default router log, change your *syslog.conf* files to include the following:

```
local7.debug                    /var/log/rtrlog
```

Use caution when enabling the remote storing of debug messages. Debug messages can quickly overwhelm your server if not properly enabled on the router.

Top

Recipe 18.9 Setting the IP Source Address for Syslog Messages

18.9.1 Problem

You want the router to use a particular source IP address for syslog messages.

18.9.2 Solution

Use the *logging source-interface* configuration command to specify a particular IP address for syslog messages:

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#logging 172.25.1.1
Router(config)#logging source-interface loopback0
Router(config)#end
Router#
```

18.9.3 Discussion

When you enable logging to a remote server, that server will normally see the source of the message as being the router's nearest interface. However, this is not always meaningful. Sometimes you want it to be a *loopback* address so that all messages from this router look the same. For example, it is a common practice to populate DNS with only the *loopback* IP addresses to facilitate router access. This means that none of the other router interfaces can be resolved using DNS:

```
Apr  2 20:27:01 172.25.2.6 94: %SYS-5-CONFIG_I: Configured from on vty0
Apr  2 20:27:48 Boston 95: %SYS-5-CONFIG_I: Configured from on vty0
```

This example shows two identical log messages originating from the same router, as they appear on the syslog server. The first message uses the IP address of a serial interface that the syslog server is unable to resolve. Note that the server still stores the message, but it uses the IP address to identify the source.

The second log message occurs after configuring the router to use the *loopback* interface as the source address. The syslog server is now able to resolve the source IP address and identifies the source as the router Boston. This makes parsing the log file for all syslog messages that belong to this router straightforward and simple.

18.9.4 See Also

Recipe 18.5

Top

Recipe 18.10 Logging Router Syslog Messages in Different Files

18.10.1 Problem

You want the Unix server to send router log messages to a different log file than the local system messages.

18.10.2 Solution

To stop router syslog messages from inundating your local system log files, use the following commands:

```
local7.info                                                    /var/log/rtrlog
*.err;kern.debug;auth.notice;mail.crit;local7.none              /var/log/syslog
*.notice;kern.debug;auth.info;mail.crit;local7.none             /var/adm/messages
```

18.10.3 Discussion

Most common syslog facilities will log all messages of a certain severity and higher (see Table 18-1) to their generic local system log files by default. However, once you configure yours*yslog.conf* file to store all router log messages to a specific file (in this case, */var/log/rtrlog*), you probably want to stop this from occurring. Suppressing router log messages from being stored in your local system log files prevents these messages from cluttering your log files with redundant data.

The first line of the previous example causes the syslog facility to store all router logs to a certain file. However, syslog will continue to archive router messages into the general system log files as well. Note that lines 2 and 3 include a wildcard option (*.*err* and *.*info*). These wildcards match the *local7* logging facility (and all others as well), causing all log messages of a certain severity level and higher to be stored. To prevent the router logs from being stored in the general system log files, use the *local7.none*, which will override the wildcard setting.

18.10.4 See Also

Recipe 18.6

Top

Recipe 18.11 Maintaining Syslog Files on the Server

18.11.1 Problem

You want to automatically rotate and archive router log files on a Unix server.

18.11.2 Solution

The Bourne shell script given in Example 18-1 will automatically rotate router log files to ensure that these files don't become too big and cumbersome to navigate. The script is intended to be invoked via a *cron* job on a daily basis, but you can also run it manually. By default, the script retains seven days' worth of archived log files and compresses files older than two days. No arguments are required or expected.

Example 18-1. rotatelog.sh

```
#!/bin/sh
#
#  rotatelog.sh -- a script to rotate log files and
#                  compress archived files
#
# Set behavior
SYSLOGPID=/etc/syslog.pid
LOGDIR=/var/log
LOG=rtrlog
DAYS=7
COMPRESS="/usr/bin/compress -f"
#
# Program body
[ -f $SYSLOGPID ] || echo "Syslog PID file doesn't exist"
if [ -d $LOGDIR ]; then
  cd $LOGDIR
  [ -f $LOG.1 ] && `$COMPRESS $LOG.1` && sleep 1
  while [ $DAYS -gt 1 ]
  do
        LOW=`expr $DAYS - 1`
        [ -f $LOG.$LOW.Z ] && mv $LOG.$LOW.Z $LOG.$DAYS.Z
        DAYS=$LOW
  done
[ -f $LOG ] || echo "Log file $LOG doesn't exist"
[ -f $LOG ] && mv $LOG $LOG.1
touch $LOG
```

```
chmod 644 $LOG
sleep 10
kill -HUP `cat $SYSLOGPID`
#
else
echo "Log directory $LOGDIR is not valid"
fi
```

18.11.3 Discussion

If left unchecked, the router log files grow until you run out of disk space. This script is designed to rotate router log files on a daily basis to ensure that they don't grow too large.

This script will rotate logs on a daily basis and retain seven days worth of archived log files, overwriting files that are older than this. To reduce the required disk space of the archived log files, the script will retain the previous day's log in a normal format, and compress the remaining days. The number of archived days stored by the script is completely configurable. For instance, to change the number of archived days to 30, alter the*DAYS* variable:

```
DAYS=30
```

The script is configured to rotate a file called */var/log/rtrlog*, but it can easily be modified to rotate any log file by changing the *LOGDIR* and *LOG* variables. *LOGDIR* contains the directory that the log files reside in, while *LOG* contains the name of the log file itself. For instance, if you want to change the script to rotate a file called */var/adm/nmslog*, modify the following lines in the script:

```
LOGDIR=/var/adm
LOG=nmslog
```

The final variable that may require modification is the *SYSLOGPID* variable. This variable contains the location of your system's *syslog.pid* file. The script assumes that the process ID number (PID), which is carried by the *SYSLOGPID* variable, can be found in the file */etc/syslog.pid*. This is a common location for Solaris-based machines (another common location is */var/run/syslog.pid*). Configuring the script with the correct *syslog.pid* location is vitally important—your log files will not rotate correctly if it is misconfigured. To find the location of your *syslog.pid* file on your system, use the following command (which can take several minutes to run):

```
server% find / -name syslog.pid -print
/etc/syslog.pid
server%
```

As mentioned, the script is intended to be launched from *cron* on a nightly basis. Since it will likely require root privileges to create the necessary files, launch the script from root's *crontab*. Below is an example of the *crontab* entry required to launch the script each day at midnight (assuming it is located in */usr/local/bin*):

```
0 0 * * *               /usr/local/bin/rotatelog.sh
```

Finally, since archived files older than one day are compressed, we should mention some methods of working with compressed files. To view a compressed file, use the *zcat* command or *uncompress -c* command. To permanently uncompress a compressed file, use the *uncompress* command (without any switches).

Top

Recipe 18.12 Testing the Syslog Sever Configuration

18.12.1 Problem

You want to test the configuration of your syslog server to ensure that the log messages are stored in their correct location.

18.12.2 Solution

The Bourne shell script in Example 18-2 will emulate syslog messages at various severity levels to ensure that your server routes them to the correct location. The script will emulate syslog messages to the *local7* syslog facility by default, but the logging facility is completely configurable. No arguments are required or expected.

Example 18-2. testlog.sh

```
#!/bin/sh
#
#    testlog.sh -- a script to test the syslog facility to ensure that
#                  messages, at various levels, are being forwarded
#                  to the correct file(s)
#
# Set Behavior
FACILITY=local7
LOGGER="/usr/bin/logger"
#
$LOGGER -p $FACILITY.emerg    "This meassage was sent to $FACILITY.emerg (0)"
$LOGGER -p $FACILITY.alert    "This meassage was sent to $FACILITY.alert (1)"
$LOGGER -p $FACILITY.crit     "This meassage was sent to $FACILITY.crit (2)"
$LOGGER -p $FACILITY.err      "This meassage was sent to $FACILITY.err (3)"
$LOGGER -p $FACILITY.warning "This meassage was sent to $FACILITY.warning (4)"
$LOGGER -p $FACILITY.notice   "This meassage was sent to $FACILITY.notice (5)"
$LOGGER -p $FACILITY.info     "This meassage was sent to $FACILITY.info (6)"
$LOGGER -p $FACILITY.debug    "This meassage was sent to $FACILITY.debug (7)"
```

18.12.3 Discussion

This script is designed to test the syslog server configuration to ensure that router log messages forward to the correct file(s). Basically, the script emulates router log messages at the various severity levels to verify how the syslog daemon handles them.

We use the Unix *logger* command to generate log messages and forward them to the syslog daemon. The server should route these log messages to same location as the router log messages. If the test log messages do not show up in the expected file or show up in undesirable locations, you should look for configuration problems in your *syslog.conf* file.

As noted, the script's default syslog facility is set to *local7*, but you can change this if necessary. For instance, if your routers are set to use *local6* (as in Recipe 18.7), the variable *FACILITY* should be set to *local6*:

```
FACILITY=local6
```

If your *syslog.conf* file includes an entry to forward *local7.info* log messages to a file called */var/log/rtrlog* (as in Recipe 18.6), the output from the script would look like this:

```
Freebsd# ./testsyslog.sh

Message from syslogd@localhost at Sun Mar 31 22:44:09 2002 ...
localhost This message was sent to local7.emerg (0)
Freebsd# tail /var/log/rtrlog
Mar 31 22:44:09 localhost This message was sent to local7.emerg (0)
Mar 31 22:44:09 localhost This message was sent to local7.alert (1)
Mar 31 22:44:09 localhost This message was sent to local7.crit (2)
Mar 31 22:44:09 localhost This message was sent to local7.err (3)
Mar 31 22:44:09 localhost This message was sent to local7.warning (4)
Mar 31 22:44:09 localhost This message was sent to local7.notice (5)
Mar 31 22:44:09 localhost This message was sent to local7.info (6)
Freebsd#
```

Note that one of the messages produced by the script was sent directly to the screen. This is because the test server's *syslog.conf* file forwards all *emergency* level syslog messages to all TTY terminals, which is a common configuration on Unix machines. Although this message will not cause any system problems, it can strike fear into other active users, so be aware.

The second part of the output shows the contents of the */var/log/rtrlog* file. The output shows seven lines of progressively decreasing priority log messages but it does not display a severity 7 (debugging) message. This is because the *syslog.conf* configuration only included a line for *local7.info*. Because the info severity is higher than debug, this configuration command does not affect debug level messages.

Finally, with a minor modification to your *syslog.conf* file, you can utilize this script to test remote syslog servers:

```
local7.info                                    @nms.oreilly.com
```

With this change, the syslog program will forward all *local7* log messages to a remote syslog server called *nms.oreilly.com*. Note that the syntax of this line introduces the @ sign to signify that a server name follows. Running the script again would forward *local7* log messages to the remote server, which would effectively emulate router log messages and test the server's syslog configuration. When testing

is completed, make sure to remove or comment out the above configuration line. Otherwise, this machine will continue to forward incoming *local7* log messages to the remote syslog server.

18.12.4 See Also

Recipe 18.6; Recipe 18.7

Top

Recipe 18.13 Preventing the Most Common Messages from Being Logged

18.13.1 Problem

You want to prevent the router from sending link up/down syslog messages for unimportant router interfaces.

18.13.2 Solution

Use the *no logging event* configuration commands to disable the logging of common interface-level messages:

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#interface Serial0/0
Router(config-if)#no logging event link-status
Router(config-if)#no logging event dlci-status-change
Router(config-if)#no logging event subif-link-status
Router(config-if)#end
Router#
```

18.13.3 Discussion

By default, log messages are sent whenever a router interface status changes states. Generally, you want to see log messages that indicate that an interface status has changed, but there are times when it can be useful to disable these types of messages. For instance, dial interfaces may cycle up and down many times throughout the course of a normal day without being cause for concern. Suppressing these messages helps keep logs uncluttered and can prevent network management staff from wasting time responding to unnecessary trouble reports:

```
%LINK-3-UPDOWN: Interface Serial0, changed state to down
%LINEPROTO-5-UPDOWN: Line protocol on Interface Serial0, changed state to down
%LINK-3-UPDOWN: Interface Serial0, changed state to up
%LINEPROTO-5-UPDOWN: Line protocol on Interface Serial0, changed state to up
```

This example shows the log messages that are sent when a router interface changes states from up to down and back to up. The *no logging event link-status* command will prevent the router from creating these messages.

On Frame Relay interfaces, DLCI state changes trigger the router to create log messages. In large

Frame Relay-based networks, many DLCI changes can occur daily, which can clutter logs and open duplicate trouble reports. The *no logging event dlci-status-change* configuration command will prevent these log messages from being created:

```
%FR-5-DLCICHANGE: Interface Serial0 - DLCI 50 state changed to INACTIVE
%LINEPROTO-5-UPDOWN: Line protocol on Interface Serial0.1, changed state to down
%FR-5-DLCICHANGE: Interface Serial0 - DLCI 50 state changed to ACTIVE
%LINEPROTO-5-UPDOWN: Line protocol on Interface Serial0.1, changed state to up
```

You can also suppress subinterface link up or down messages by using the *no logging event subif-link-status* configuration command. The following example shows a typical Frame Relay interface failure. Note that the router sends a "line protocol down" log message for the main interface and each of the subinterfaces. Although this example only shows one subinterface, it is not uncommon to see dozens of subinterfaces on a single physical interface. In these cases, it can be useful to suppress subinterface messages:

```
%LINK-3-UPDOWN: Interface Serial0, changed state to down
%FR-5-DLCICHANGE: Interface Serial0 - DLCI 50 state changed to DELETED
%FR-5-DLCICHANGE: Interface Serial0 - DLCI 102 state changed to DELETED
%FR-5-DLCICHANGE: Interface Serial0 - DLCI 103 state changed to DELETED
%LINEPROTO-5-UPDOWN: Line protocol on Interface Serial0.1, changed state to down
%LINEPROTO-5-UPDOWN: Line protocol on Interface Serial0, changed state to down
```

Top

Recipe 18.14 Rate-Limiting Syslog Traffic

18.14.1 Problem

You wish to rate-limit the syslog traffic to your server.

18.14.2 Solution

Use the *logging rate-limit* configuration command to limit the number of syslog packets sent to your server:

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#logging 172.25.1.1
Router(config)#logging rate-limit 30 except warnings
Router(config)#end
Router#
```

To rate-limit the number of log messages sent to the console port, use the following command:

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#logging rate-limit console 25 except warnings
Router(config)#end
Router#
```

This feature became available starting in IOS Version 12.1(3)T.

18.14.3 Discussion

By default, a router that is configured for remote logging will forward all log messages to the syslog server as they are created, regardless of how many messages there are. The*rate-limit* command will throttle the number of packets to ensure that router won't flood the network or syslog server. It is particularly useful to throttle syslog messages when forwarding debug traces or if the network is congested.

Cisco provides the option to throttle log messages sent to the console port as well. This feature is important since all messages written to the console port cause CPU interrupts. If a large number of log messages are being sent to the console port, the router can suffer noticeable service degradation. Being

able to rate-limit messages is an effective alternative to completely disabling them.

The syntax for rate-limiting includes several options. The first example limits the rate of syslog messages to 30 messages per second. You can configure this option to send any number from 1 to 10,000 messages per second. Since log messages vary in length, it is difficult to calculate a meaningful number in terms of bytes per second. However, a typical average size for a log message is between 150 and 170 bytes. So we can roughly estimate that 30 messages per second will correspond to 36,000 to 40,800 bits per second, which is a good limit for serial lines.

Both examples in this section use optional the keyword *except*. Use this keyword to ensure that only noncritical messages become rate-limited. For example, to rate-limit all messages at a warning severity level or lower, and to allow all higher severity messages to be sent without restriction, use the *except* keyword. Note that the keyword *all* is equivalent to setting the *except* option at the debug level, meaning all messages are rate-limited.

Top

Chapter 19. Access Lists

Top

Introduction

An Access Control List (ACL) is generically a method for doing pattern matching based on protocol information. There are many reasons for doing this type of pattern matching, such as restricting access for security reasons or restricting routing tables for performance reasons.

Cisco has several different general kinds of access lists. The most common are the numbered ACLs, which are summarized in Table 19-1. But there are also named access lists, reflexive access lists, context-based access lists, and rate-limit access lists. There are even timed access lists that can have different effects at different times of day, although we will not cover them in this book. Within each of these general categories, there are many different types of ACLs that match on different protocol information. When working with route filtering, it is often easiest to work with prefix lists, which are another type of ACL discussed in more detail in Chapter 6, Chapter 7, Chapter 8 and Chapter 9.

You can apply an ACL in many different ways. Applied to an interface, you can use it to accept or reject incoming or outgoing packets based on protocol information such as source or destination address, port number, protocol number, and so forth. Applied to a routing protocol, this same ACL might prevent the router from sharing information about a particular route. And, applied to a route map, the ACL could just identify packets that need to be tagged or treated differently.

Table 19-1 shows all of the current ranges for numbered access lists. Cisco periodically adds new ranges to this list, so earlier IOS levels may not support all of these ACL types. Also bear in mind that if your IOS feature set doesn't support a particular protocol such as IPX, XNS, or AppleTalk, the corresponding ACL type will also be unavailable.

Table 19-1. Numbered access list types

Numeric range	Access list type
1 - 99	Standard IP ACL
100 - 199	Extended IP ACL
200 - 299	Ethernet type code ACL
300 - 399	DECNET ACL
400 - 499	XNS ACL
500 - 599	Extended XNS ACL

Numeric range	Access list type
600 - 699	AppleTalk ACL
700 - 799	48-bit MAC address ACL
800 - 899	IPX ACL
900 - 999	Extended IPX ACL
1000 - 1099	IPX service advertisement protocol
1100 - 1199	Extended 48-bit MAC address ACL
1200 - 1299	IPX NLSP ACL
1300 - 1999	Standard IP ACL, expanded range
2000 - 2699	Extended IP ACL, expanded range
2700 - 2999	SS7 (Voice) ACL

Clearly many of these ranges deal with protocols or technologies that are beyond the scope of this book. This book's primary focus is on IP-based technologies, so this chapter will not discuss ACL types that are intended for use with other protocols.

A named ACL is really just another way of writing either a standard or extended IP ACL. Named ACLs can make your configuration files considerably easier to read. Some commands that use an ACL for pattern matching will not accept named ACLs, but, for the most part, named ACLs are interchangeable with normal IP ACLs. Their chief advantage is that you can nest other ACLs inside of a named ACL for greater flexibility.

Reflexive ACLs are more sophisticated objects that can contain temporary entries. Reflexive ACLs need to be nested inside of named ACLs. A reflexive ACL has two parts that are generally nested in two different named ACLs. One part watches for packets of a particular type using normal extended IP ACL syntax. As soon as the router sees this packet, it activates a matching rule in another ACL. This allows you to do things like permitting inbound traffic of a particular type only after the router sees an initial outbound packet of the matching type.

However, reflexive ACLs are somewhat limited in their scope because they are not able to read into the payloads of IP packets. Many applications have more complicated behavior, such as using dynamically generated port numbers. To handle this type of situation, Cisco has developed another type of ACL called Context-Based Access Control (CBAC).

Like reflexive ACLs, CBAC works by turning on and off temporary access list rules. However, CBAC actively monitors applications using a stateful inspection algorithm that allows the router to react to the application and dynamically create new ACL rules. It can also watch for unusual application behavior

and dynamically disable the corresponding temporary ACL rules.

You should always remember that every ACL ends with an implicit *deny all* clause. This means that if you are matching items (packets, for example) with an ACL, and the item fails to match any of the explicitly listed clauses of the ACL and falls off the end, it is the same as if the item matched an explicit *deny* clause. For this reason, if you are trying to block certain unwanted packets (for example), but want to allow all others to pass, you must remember to include a *permit all* statement at the end of the ACL.

For more information on ACLs in general, refer to Cisco IOS Access Lists (O'Reilly).

Top

Recipe 19.1 Filtering by Source or Destination IP Address

19.1.1 Problem

You want to block packets to or from certain IP addresses.

19.1.2 Solution

You can use standard access lists to block packets from specified IP source addresses:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#access-list 50 deny host 10.2.2.2
Router1(config)#access-list 50 permit any
Router1(config)#interface Serial0/1
Router1(config-if)#ip access-group 50 in
Router1(config-if)#end
Router1#
```

You can filter packets based on both the source and destination addresses with an extended access list:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#access-list 150 deny ip host 10.2.2.2 host 172.25.25.1
Router1(config)#access-list 150 permit ip any any
Router1(config)#interface Serial0/1
Router1(config-if)#ip access-group 150 in
Router1(config-if)#end
Router1#
```

19.1.3 Discussion

The most obvious use for access lists is traffic filtering. The two examples in this recipe both show how to use access control lists for filtering inbound packets. The first example uses the following access list:

```
Router1(config)#access-list 50 deny host 10.2.2.2
Router1(config)#access-list 50 permit any
```

This is a numbered ACL with a value between 1 and 99, making it a *standard access list.* Using standard ACLs like this allows you to filter based only on the source IP address. In the example, we have chosen

to deny a single host address, `10.2.2.2`. All other packets are permitted. This is a somewhat artificial thing to do, of course. It is more likely that you would want to allow only a limited group of devices and block all of the others. For example, you might want to allow all of the devices on `10.2.2.0/24`, except the host `10.2.2.2`, and drop packets from any other devices. You could do this with the following standard ACL:

```
Router1(config)#access-list 50 deny host 10.2.2.2
Router1(config)#access-list 50 permit 10.2.2.0 0.0.0.255
Router1(config)#access-list 50 deny any
```

The order of the statements in an access list is critical. Consider what would have happened if we had put a line that permitted `10.2.2.0/24` before the line that denied the specific host (`10.2.2.2`). Then, each time the router receives a packet from the forbidden host, it compares it to the ACL. Because this host is part of the permitted range, the router would accept the packet and not look any further.

Also, because the router will stop processing an ACL as soon as it finds a match, you can save the processor a lot of work by putting the most common rules near the top. You want the router to find a match as early as possible while parsing the ACL. All of the examples in this chapter are relatively short, but if you have an ACL that is several hundred lines long, good ordering so that most of your packets match early lines in the ACL can make a significant improvement in processing time. Ordering will affect router CPU load and can also improve jitter and latency issues.

The other important thing to look out for when constructing an ACL like this is the fact that they use wildcard bits (rather than netmask format) when defining ranges of addresses. In this example, we wanted to specify the range `10.2.2.0/24`. This means that we want to fix all of the bits in the first three octets of the address and allow any pattern of bits in the last octet. We can achieve this with a wildcard pattern of `0.0.0.255`. Refer to Recipe 5.3 for a more detailed discussion of the differences between wildcards and netmasks.

The basic notation is an IP address followed by a wildcard pattern. But there are two special cases. Suppose you want to match on a single IP address for a particular device, such as `10.2.2.2`. You can write this as follows:

```
Router1(config)#access-list 50 deny 10.2.2.2 0.0.0.0
```

This wildcard pattern means that there are no free bits in the address, so we are looking for an exact match. Alternatively, you can use the more intuitive version that we used earlier in this recipe, with the *host* keyword:

```
Router1(config)#access-list 50 deny host 10.2.2.2
```

The two forms have an identical effect. In fact, for standard IP ACLs like this one, the router will replace both of these with the following:

```
Router1(config)#access-list 50 deny 10.2.2.2
```

The *host* keyword is not redundant, however, for extended IP ACLs, which we will discuss later.

Alternatively, if you want to match any device, you could write this:

```
Router1(config)#access-list 50 deny 0.0.0.0 255.255.255.255
```

Because all of the wildcard bits in this pattern are set, it means that any bit may have a value of either 0 or 1. Therefore, this pattern will match any IP address. The router will rewrite this ACL with the following simpler form:

```
Router1(config)#access-list 50 deny any
```

As we mentioned in the introduction to this chapter, there are many different ways to apply an ACL. In this case, we want to apply this ACL to filter IP packets received on a particular router interface. To do so, we use the *ip access-group* command on the interface that will receive the data stream that we want to filter:

```
Router1(config)#interface Serial0/1
Router1(config-if)#ip access-group 50 in
```

In this command, the keyword *in* tells the router to apply the filter to inbound packets, that is, packets received by this interface. In some cases, you might want to filter outbound packets instead. The router will even let you apply a different ACL to inbound and outbound packets:

```
Router1(config)#interface Serial0/1
Router1(config-if)#ip access-group 50 in
Router1(config-if)#ip access-group 51 out
```

The outbound *access-group* command has an interesting quirk that you should be aware of. This command will not filter packets that originate on the router itself. This is important because if you apply an outbound filter and then try to test it with a *ping* or telnet from the router, your rule will appear not to work. Make sure to do these tests from a downstream device.

It's also important to remember that the router originates packets in several less obvious situations. For example, if you are using tunnels, the tunnel packets themselves originate on the router—even if the router is encapsulating IP packets from downstream devices. The same is true when the router acts as a gateway for other protocols, such as SNA or X.25.

This also has security implications: if somebody can convince the router to originate packets for them, they will bypass the rule. So, while the outbound access-group command is extremely useful, you need to be careful about how you use it, or it might be less effective than you expect.

The second example in the solution section of this recipe does a similar kind of filtering, except that this time we have used an extended ACL:

```
Router1(config)#access-list 150 deny ip host 10.2.2.2 host 172.25.25.1
Router1(config)#access-list 150 permit ip any any
```

The syntax of the extended IP ACL is somewhat more involved than that of the standard ACL. It includes the same *permit* or *deny* keywords as the standard IP ACL. The next keyword in this example selects the IP protocol. Refer to Recipe 19.3 for a discussion of other options.

Extended IP ACLs always include two IP addresses. The first is the source address, and the second is the destination. In the first line of the example, we have specified two host addresses for the source and destination, while the second line matches any address in either the source or the destination fields of the IP packet.

You can use the same wildcard matching systems for IP addresses that we discussed earlier. Suppose, for example, that we want to allow any device on the `10.2.2.0/24` segment to access any destination device, but we want to specifically prevent the host, `10.2.2.2` from reaching `172.25.25.1`, and we want to drop all packets from any other devices. We could do this with the following extended IP ACL:

```
Router1(config)#access-list 150 deny ip host 10.2.2.2 host 172.25.25.1
Router1(config)#access-list 150 permit ip 10.2.2.0 0.0.0.255  any
Router1(config)#access-list 150 deny ip any any
```

Note that the last line explicitly denies everything that wasn't matched in one of the previous lines. This is actually not necessary because every ACL implicitly ends with a*deny all*, whether you include it or not. However, including an explicit *permit all* or *deny all* clause at the end of an ACL is generally a good practice because it makes things more clear. Also, when you look at an ACL with the *show access-list* command, it gives you a breakdown of how many times each rule found a match:

```
Router1#show access-list 150
Extended IP access list 150
    deny ip host 10.2.2.2 host 172.25.25.1 (422 matches)
    permit ip 10.2.2.0 0.0.0.255 any (743 matches)
    deny ip any any (387 matches)
Router1#
```

Including an explicit *deny all* rule at the end of this ACL allows you to tell exactly how many packets were affected.

As we discuss in Recipe 19.8, you can configure this line to log exactly which packets reach all the way to the end of the ACL without matching one of the earlier rules.

Finally, we need to mention one of the most annoying problems involved in managing large ACLs. If you add a new rule to an ACL, it will always go at the very end of the list. This is not always what you want. Worst of all, there is no way to remove individual lines from an ACL without removing the entire list.

We have found that the easiest way to get around this problem is to use the *show running-config* command to get a complete listing of the ACL you want to edit. Copy this into a text editor of some kind on your local workstation. Then you can use any common text editor to create the modified ACL. Delete the old ACL and copy in the new one. As we mentioned in Chapter 1, if you use TFTP to

transfer the new commands into the router, it will not make the change until the transfer is complete. This is particularly important for ACLs, where you could enter the first line and then find yourself locked out of the router by the implicit *deny all* that follows it.

19.1.4 See Also

Recipe 5.3; Recipe 19.3; Recipe 19.8

Top

Recipe 19.2 Adding a Comment to an ACL

19.2.1 Problem

You want to add a human-readable comment to an ACL to help other engineers understand what you have done.

19.2.2 Solution

You can add a comment to any standard or extended IP ACL using the *remark* keyword:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#access-list 50 remark Authorizing thy trespass with compare
Router1(config)#access-list 50deny host 10.2.2.2
Router1(config)#access-list 50 permit 10.2.2.0 0.0.0.255
Router1(config)#access-list 50 permit any
```

19.2.3 Discussion

This command can be quite useful when you have to keep track of many different ACLs on a router, particularly when several of them look similar. The comment field can be up to 100 characters long. If you require more space, simply add more remark lines to the ACL:

```
Router1(config)#access-list 50 remark Authorizing thy trespass with compare
Router1(config)#access-list 50remark My self corrupting salving thy amiss,
Router1(config)#access-list 50remark Excusing thy sins more than thy sins are
Router1(config)#access-list 50 remark Shakespeare, Sonnet 35
```

When you display this ACL using the *show access-list* command, it will not show the remark lines:

```
Router1#show access-list 50
Standard IP access list 50
    deny   10.2.2.2
     permit 10.2.2.0, wildcard bits 0.0.0.255
    permit any
Router1#
```

The only way to see these comments is to look at the router's configuration file:

```
Router1#show running-config | include access-list 50
access-list 50 remark Authorizing thy trespass with compare
```

```
access-list 50 remark My self corrupting salving thy amiss,
access-list 50 remark Excusing thy sins more than thy sins are
access-list 50 remark Shakespeare, Sonnet 35
access-list 50 deny   10.2.2.2
access-list 50 permit 10.2.2.0 0.0.0.255
access-list 50 permit any
access-list 50 remark
Router1#
```

Note that the router does not reorder the remark lines in the ACL, so you can use this feature to explain line-by-line what each command does:

```
Router1(config)#access-list 50 remark loathsome canker
Router1(config)#access-list 50 deny host 10.2.2.2
Router1(config)#access-list 50 remark sweetest bud
Router1(config)#access-list 50 permit 10.2.2.0 0.0.0.255
Router1(config)#access-list 50 permit any
```

19.2.4 See Also

Complete Sonnets, William Shakespeare (Dover)

Top

Recipe 19.3 Filtering by Application

19.3.1 Problem

You want to filter access to certain applications.

19.3.2 Solution

Extended IP access lists can also filter based on application information such as protocol and port numbers:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#access-list 151 permit tcp any any eq www
Router1(config)#access-list 151 deny tcp any any gt 1023
Router1(config)#access-list 151 permit icmp any any
Router1(config)#access-list 151 permit udp any any eq ntp
Router1(config)#access-list 151 deny ip any any
Router1(config)#interface Serial0/1
Router1(config-if)#ip access-group 151 in
Router1(config-if)#end
Router1#
```

19.3.3 Discussion

This example shows how to construct an extended IP ACL to filter traffic based on applications. In Recipe 19.1, we showed how to use extended IP ACLs to match on any combination of source and/or destination IP addresses. But the extended IP ACL also allows you to match on just about anything in the IP packet header.

The first argument after the *permit* or *deny* keyword represents the IP protocol type:

```
Router1(config)#access-list 151 permit tcp any any eq www
```

In this case, we want to match a TCP-based application, so we have used the keyword *tcp* in this position. This field represents the IP protocol number, which is an 8-bit value. TCP is protocol number 6, UDP is 17, and ICMP uses protocol number 1. The IANA has registered 134 different protocol numbers. You can find the complete list of registered IP protocols online at http://www.iana.org/assignments/protocol-numbers. Cisco supplies helpful mnemonics for several of these protocols, such as the *tcp*, *udp*, and *icmp* keywords used in the example, so you don't have to

remember the protocol numbers. Table 19-1Table 19-1 shows all of the IP protocols for which Cisco supplies mnemonic keywords. You can always use the protocol number in decimal form if you prefer, but the router will replace it with the mnemonic (if one exists) in its configuration file.

Table 19-2. IP protocol numbers and their extended ACL keywords

Protocol number	Keyword	Description
1	icmp	Internet Control Message Protocol
2	igmp	Internet Gateway Message Protocol
4	ipinip	IP-in-IP tunnel protocol
6	tcp	Transmission Control Protocol
9	igrp	Interior Gateway Routing Protocol
17	udp	User Datagram Protocol
21	nos	KA9Q tunnel protocol
47	gre	Generic Routing Encapsulation tunnel protocol
50	esp	IPSec Encapsulation Security Payload
51	ahp	IPSec Authenticating Header Protocol
88	eigrp	Enhanced Interior Gateway Routing Protocol
89	ospf	Open Shortest Path First routing protocol
103	pim	Protocol Independent Multicast protocol
108	pcp	IP Payload Compression Protocol

As we showed in Recipe 19.1, you can match on any IP protocol number by simply using the keyword *ip*.

The source and destination IP addresses come after the IP protocol number or keyword. We described how to use these fields in Recipe 19.1. Remember that the address keyword *any* is shorthand that stands for an address of `0.0.0.0` with a wildcard pattern of `255.255.255.255`.

Following each address is an optional field in which you can specify particular protocol information such as port numbers. In the following example, we match on TCP port 80, which is used by the HTTP protocol.

There are many possible mnemonic keywords for different TCP port numbers:

```
Router1(config)#access-list 151 permit tcp any eq ?
  <0-65535>    Port number
  bgp          Border Gateway Protocol (179)
  chargen      Character generator (19)
  cmd          Remote commands (rcmd, 514)
  daytime      Daytime (13)
  discard      Discard (9)
  domain       Domain Name Service (53)
  echo         Echo (7)
  exec         Exec (rsh, 512)
  finger       Finger (79)
  ftp          File Transfer Protocol (21)
  ftp-data     FTP data connections (20)
  gopher       Gopher (70)
  hostname     NIC hostname server (101)
  ident        Ident Protocol (113)
  irc          Internet Relay Chat (194)
  klogin       Kerberos login (543)
  kshell       Kerberos shell (544)
  login        Login (rlogin, 513)
  lpd          Printer service (515)
  nntp         Network News Transport Protocol (119)
  pim-auto-rp  PIM Auto-RP (496)
  pop2         Post Office Protocol v2 (109)
  pop3         Post Office Protocol v3 (110)
  smtp         Simple Mail Transport Protocol (25)
  sunrpc       Sun Remote Procedure Call (111)
  syslog       Syslog (514)
  tacacs       TAC Access Control System (49)
  talk         Talk (517)
  telnet       Telnet (23)
  time         Time (37)
  uucp         Unix-to-Unix Copy Program (540)
  whois        Nicname (43)
  www          World Wide Web (HTTP, 80)

Router1#
```

As with the IP protocol numbers listed in Table 19-1, you can substitute the decimal numerical value for any of these keywords, and the router will replace it with the keyword. For any other applications not included in this list, you must use the decimal port number.

In our example, the mnemonic for port 80 is *www*:

```
Router1(config)#access-list 151 permit tcp any any eq www
```

Note that the keywords *eq www* appear after the destination IP address, rather than the source IP address. This is because we are looking for the destination TCP port number. If you need to match on a source port number instead, you could simply move these keywords to follow the source IP address:

```
Router1(config)#access-list 151 permit tcp any eq www any
```

Of course, you can always match on both:

```
Router1(config)#access-list 151 permit tcp any eq www any eq www
```

Note, however, that this ACL will score a correct match only if both source and destination TCP port numbers match. If you wanted to match HTTP traffic between any two devices, but didn't know which device had initiated the TCP session, you would need to include two separate lines like this:

```
Router1(config)#access-list 151 permit tcp any any eq www
Router1(config)#access-list 151 permit tcp any eq www any
```

The IANA reserves the TCP port numbers 1024 and above for local and temporary applications. Many TCP implementations use these high-numbered ports for source port numbers, and for temporary or ephemeral purposes. So it is relatively common to see ACLs that restrict the use of these ports. We included an example ACL rule in this recipe:

```
Router1(config)#access-list 151 deny tcp any any gt 1023
```

This command will block all packets that have a destination port number greater than 1023 (that is, ports 1024 through 65,535). Remember that TCP applications often use these high port numbers for source ports, so you need to be careful about traffic direction when you apply such an ACL.

There is a similar set of port numbers for UDP applications:

```
Router1(config)#access-list 151 permit udp any eq ?
  <0-65535>    Port number
  biff         Biff (mail notification, comsat, 512)
  bootpc       Bootstrap Protocol (BOOTP) client (68)
  bootps       Bootstrap Protocol (BOOTP) server (67)
  discard      Discard (9)
  dnsix        DNSIX security protocol auditing (195)
  domain       Domain Name Service (DNS, 53)
  echo         Echo (7)
  isakmp        Internet Security Association and Key Management Protocol (500)
  mobile-ip    Mobile IP registration (434)
  nameserver    IEN116 name service (obsolete, 42)
  netbios-dgm   NetBios datagram service (138)
  netbios-ns    NetBios name service (137)
  netbios-ss    NetBios session service (139)
  ntp          Network Time Protocol (123)
  pim-auto-rp   PIM Auto-RP (496)
  rip          Routing Information Protocol (router, in.routed, 520)
  snmp         Simple Network Management Protocol (161)
  snmptrap     SNMP Traps (162)
  sunrpc       Sun Remote Procedure Call (111)
  syslog       System Logger (514)
  tacacs       TAC Access Control System (49)
  talk         Talk (517)
```

```
  tftp           Trivial File Transfer Protocol (69)
  time           Time (37)
  who            Who service (rwho, 513)
  xdmcp          X Display Manager Control Protocol (177)

Router1#
```

For example, you could block all Sun RPC traffic, which includes important but chatty applications such as Network File System (NFS):

```
Router1(config)#access-list 151 deny udp any eq sunrpc any
Router1(config)#access-list 151 deny udp any any eq sunrpc
```

This will block RPC traffic going in either direction because we applied the UDP port number file separately to the source and destination ports.

Like TCP, UDP port number values from 1024 through 65,535 are often used for temporary purposes such as source port numbers. You can control the use of these port numbers with a similar rule:

```
Router1(config)#access-list 151 deny udp any any gt 1023
```

You can also create an access list to look for specific types of ICMP messages in exactly the same way that we matched on TCP and UDP port numbers. The difference for ICMP message types is that each ICMP packet has only a single type field, rather than a source and destination port. You specify the type after the addresses on an Extended ACL, as follows:

```
Router1(config)#access-list 190 permit icmp any any echo
```

This ACL looks for ICMP echo request packets, such as those sent by the ping utility. This feature allows you to treat these *ping* packets differently than, for example, an ICMP unreachable message. In fact, this command allows you to look for any ICMP message type number between 0 and 255 either by the decimal number, or by mnemonic:

```
Router1(config)#access-list 190 permit icmp any any ?
  <0-255>                      ICMP message type
  administratively-prohibited  Administratively prohibited
  alternate-address            Alternate address
  conversion-error             Datagram conversion
  dod-host-prohibited          Host prohibited
  dod-net-prohibited           Net prohibited
  dscp                         Match packets with given dscp value
  echo                         Echo (ping)
  echo-reply                   Echo reply
  fragments                    Check non-initial fragments
  general-parameter-problem    Parameter problem
  host-isolated                Host isolated
  host-precedence-unreachable  Host unreachable for precedence
  host-redirect                Host redirect
  host-tos-redirect            Host redirect for TOS
```

```
host-tos-unreachable          Host unreachable for TOS
host-unknown                  Host unknown
host-unreachable              Host unreachable
information-reply             Information replies
information-request           Information requests
log                           Log matches against this entry
log-input                     Log matches against this entry, including input
                              interface
mask-reply                    Mask replies
mask-request                  Mask requests
mobile-redirect               Mobile host redirect
net-redirect                  Network redirect
net-tos-redirect              Net redirect for TOS
net-tos-unreachable           Network unreachable for TOS
net-unreachable               Net unreachable
network-unknown               Network unknown
no-room-for-option            Parameter required but no room
option-missing                Parameter required but not present
packet-too-big                Fragmentation needed and DF set
parameter-problem             All parameter problems
port-unreachable              Port unreachable
precedence                    Match packets with given precedence value
precedence-unreachable        Precedence cutoff
protocol-unreachable          Protocol unreachable
reassembly-timeout            Reassembly timeout
redirect                      All redirects
router-advertisement          Router discovery advertisements
router-solicitation           Router discovery solicitations
source-quench                 Source quenches
source-route-failed           Source route failed
time-exceeded                 All time exceededs
time-range                    Specify a time-range
timestamp-reply               Timestamp replies
timestamp-request             Timestamp requests
tos                           Match packets with given TOS value
traceroute                    Traceroute
ttl-exceeded                  TTL exceeded
unreachable                   All unreachables
<cr>
```

Note that not all of the range of possible ICMP message types have been officially allocated. Please refer to the Internet Assigned Numbers Authority (IANA) web site for a complete listing of standard ICMP message types: http://www.iana.org/assignments/icmp-parameters.

19.3.4 See Also

Recipe 19.1; Recipe 19.6

Top

Recipe 19.4 Filtering Based on TCP Header Flags

19.4.1 Problem

You want to filter based on the flag bits in the TCP header.

19.4.2 Solution

The following ACL blocks contain several illegal combinations of TCP header flags:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#access-list 161 deny tcp any any ack fin psh rst syn urg
Router1(config)#access-list 161 deny tcp any any rst syn
Router1(config)#access-list 161 deny tcp any any rst syn fin
Router1(config)#access-list 161 deny tcp any any rst syn fin ack
Router1(config)#access-list 161 deny tcp any any syn fin
Router1(config)#access-list 161 deny tcp any any syn fin ack
Router1(config)#end
Router1#
```

19.4.3 Discussion

There are six flag bits in the TCP header that the devices use to control the session:

ACK

: Acknowledgement.

SYN

: Synchronize sequence numbers.

FIN

: Terminate the session.

RST

: Reset the session.

PSH

: Push this data to the application immediately. This usually means that all of the data has been

sent.

URG

Look at the Urgent pointer later in the packet.

TCP uses a so-called three-way handshake to set up sessions. To start a session, the client device sends a packet with the SYN bit, which is an instruction to synchronize sequence numbers. The server device responds with a packet that has both SYN and ACK bits set, which the first device then acknowledges with an ACK to complete the handshake process.

The session teardown procedure is similar, but it actually uses four packets instead of three. One device sends the other a packet with the FIN bit set. The second device then responds to this with an ACK. Then, in a separate packet, the second device sends its own FIN, which the first device responds to with an ACK to terminate the session.

Devices use the RST flag for a few different reasons, but one of the most common is the so-called *abortive close*. This happens when one device can't wait around for the other device to acknowledge the end of the session using the normal FIN and ACK pattern. So, it simply sends a packet that has both the RST and ACK bits set to end the session. There is no need for the other device to respond to this packet.

Obviously, some combinations of these bits are not valid, however. For example, it makes no sense to have a single packet with both the SYN and FIN bits set. And, in defining test cases for TCP implementations, RFC 1025 defines a packet with all 6 bits set as a *nastygram* (or a kamikaze packet, Christmas tree packet, or lamp test segment). The first line in the example ACL blocks nastygrams:

```
Router1(config)#access-list 161 deny tcp any any ack fin psh rst syn urg
```

The remaining lines block other illegal combinations of flags.

19.4.4 See Also

RFC 1025

Top

Recipe 19.5 Restricting TCP Session Direction

19.5.1 Problem

You want to filter TCP sessions so that only the client device may initiate the application.

19.5.2 Solution

You can use the *established* keyword to restrict which device is allowed to initiate the session. In the following example, we want to allow the client device to telnet to the server, but not the other way around:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#access-list 148 permit tcp any eq telnet any established
Router1(config)#access-list 148 deny ip any any
Router1(config)#interface FastEthernet0/0
Router1(config-if)#ip access-group 148 in
Router1(config-if)#end
Router1#
```

19.5.3 Discussion

In this example, the interface will accept incoming TCP packets only if they have a TCP source port number of 23 (Telnet) and this TCP session is already established. It does not restrict the destination port number; it would be whatever random high-numbered port the initiating device had originally selected for its source port when it started the session.

The router considers an established TCP connection to be one that has either the RST or ACK bits set. We discuss these TCP header flags in more detail in Recipe 19.4. Because this does not include packets with only the SYN bit set in particular, it is impossible to create a new TCP connection.

Note that you could actually write the same thing explicitly as two rules:

```
Router1(config)#access-list 148 permit tcp any eq telnet any ack
Router1(config)#access-list 148 permit tcp any eq telnet any rst
```

The combination of these two rules is identical to the version in the example:

```
Router1(config)#access-list 148 permit tcp any eq telnet any established
```

But the version with the *established* keyword executes more efficiently. Note that putting both RST and ACK in the same rule would match packets with both RST and ACK set, not one or the other.

To see why the *established* keyword is sometimes necessary, imagine what would happen if it were not present. The interface would accept any inbound TCP packets that happened to have a source port of 23. But this could be literally anything. A moderately clever hacker who knew how to set the source port on his Telnet application could easily initiate a connection to any device on the other side of the router.

So, if you are using ACLs to control TCP applications for security reasons, you should consider using the *established* keyword.

19.5.4 See Also

Recipe 19.4

Top

Recipe 19.6 Filtering Multiport Applications

19.6.1 Problem

You want to filter an application that uses more than one TCP or UDP port.

19.6.2 Solution

This example shows how to filter both FTP control and data sessions:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#access-list 152 permit tcp any any eq ftp
Router1(config)#access-list 152 permit tcp any any eq ftp-data established
Router1(config)#interface FastEthernet0/0
Router1(config-if)#ip access-group 152 in
Router1(config-if)#end
Router1#
```

19.6.3 Discussion

Some protocols use multiple ports. A classic example is FTP, which is shown in the example. It is worthwhile to review how the FTP protocol works. For more details, please consult RFC 959.

When a client device wants to connect to a server to upload or download files, it makes a TCP connection on port 21. This port 21 connection carries all of the interactive user traffic such as usernames and passwords, as well as commands to move around to different directories. FTP also uses this control session to tell the server what port number it wants to use for transferring data. This will typically be a high-numbered temporary TCP port.

When the user wants to transfer a file, he traditionally types a put or get command on the server. We say traditionally because this is not quite how things work when your FTP client software is driven through a web browser, as we discuss in Recipe 19.12.

The server then makes a new TCP connection to the high-numbered port on the client device that it previously learned about through the control session. The source port for this connection is the well-known FTP data port number 20. This is backwards from most TCP connections in which the client device connects to the server using a well-known destination port number. Here, the server connects to the client using a well-known source port number.

The client and server exchange the file, then disconnect this FTP data connection, leaving the FTP control connection on port 21 active. The server will actually use the FTP data connection to transfer any bulk data, including directory listings as well as files. You can match both the control and data traffic streams using the ACL shown in this recipe.

In this example, we assume that the client device is connected to the router's `FastEthernet0/0` interface, perhaps through other downstream routers. And, for the sake of the example, we assume that this is the only data that we want to allow.

The router will receive a TCP packet from client device as it initiates the FTP session with destination port 21. We match this connection with the following extended IP ACL:

```
Router1(config)#access-list 152 permit tcp any any eq ftp
```

Note that we have used the keyword *ftp* in this ACL to mean TCP port 21.

Then, when there is data to exchange, the server will make a connection back to the client device on port number 20. The ACL keyword for this port is *ftp-data*:

```
Router1(config)#access-list 152 permit tcp any any eq ftp-data established
```

It's important to note that the access group is applied inbound to packets received on the client FastEthernet port. So this ACL will not apply to any of the packets sent from the server to the client device, only those sent from the client to the server. However, this is sufficient because the devices cannot establish a TCP session unless they can both send packets.

For a more generic multiport TCP application, you can specify a range of ports in the ACL with the *range* keyword:

```
Router1(config)#access-list 153 permit tcp any any range 6000 6063
```

This example matches any packets whose destination port is between 6000 and 6063 inclusive, which is the range commonly used by the X Window System. You can also specify open-ended ranges. For example, to match any TCP port number greater than 1023, use the *gt* keyword:

```
Router1(config)#access-list 153 permit tcp any any gt 1023
```

There are similar "less than" and "not equal to" operators for port numbers:

```
Router1(config)#access-list 153 permit tcp any any lt 1024
Router1(config)#access-list 153 permit tcp any any neq 666
```

As an aside, TCP port number 666 is used by the Doom interactive network game, making it an excellent candidate for filtering.

These same operations also apply identically for UDP port numbers:

```
Router1(config)#access-list 154 permit udp any any range 6000 6063
```

```
Router1(config)#access-list 155 deny udp any any gt 1023
Router1(config)#access-list 156 permit udp any any lt 1024
Router1(config)#access-list 157 permit udp any any neq 666
```

19.6.4 See Also

Recipe 19.3; Recipe 19.6; Recipe 19.12; RFC 959

Top

Recipe 19.7 Filtering Based on DSCP and TOS

19.7.1 Problem

You want to filter based on IP QoS information.

19.7.2 Solution

You can filter packets based on the contents of the Differentiated Services Control Point (DSCP) field using the *dscp* keyword:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#access-list 162 permit ip any any dscp af11
Router1(config)#end
```

Similarly, to filter based on TOS, you can use the *tos* keyword:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#access-list 162 permit ip any any tos max-reliability
Router1(config)#end
```

And you can filter based on IP Precedence using the *precedence* keyword:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z
Router1(config)#access-list 162 permit ip any any precedence flash
Router1(config)#end
```

19.7.3 Discussion

In Chapter 11 and Appendix B we discuss the DSCP, IP TOS, and IP Precedence fields in more detail. Chapter 11 also includes several examples of ACLs that filter based on this information. Please refer to these sections for more information.

The first example looks for packets that have a DSCP field value of AF11, which has a bit pattern of 001010, or a decimal value of 10. The second example matches packets with a TOS value of maximum reliability, which has a decimal value of 2.

Note that you can use the decimal numerical values for any TOS, Precedence, or DSCP field, and the

router will simply replace it with the mnemonic keyword, if one exists. For example, we could have written the second example as follows:

```
Router1(config)#access-list 162 permit ip any any tos 2
```

In this case, the router would have replaced the number 2 with the *max-reliability* keyword. However, there is no mnemonic keyword corresponding to the TOS value, 3. The router will accept values that do not have well-known names such as this, but it will leave them as numerical values in the configuration file.

19.7.4 See Also

Chapter 11; Appendix B

Top

Recipe 19.8 Logging when an Access List Is Used

19.8.1 Problem

You want to know when the router invokes an access list.

19.8.2 Solution

Access lists can generate log messages. The following example will allow all packets to pass, but will record them:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#access-list 150 permit ip any any log
Router1(config)#interface Serial0/1
Router1(config-if)#ip access-group 150 in
Router1(config-if)#end
Router1#
```

In this example, we use the *log-input* keyword to include additional information about where the packets came from:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#access-list 150 permit tcp any any log-input
Router1(config)#access-list 150 permit ip any any
Router1(config)#interface Serial0/1
Router1(config-if)#ip access-group 150 in
Router1(config-if)#end
Router1#
```

19.8.3 Discussion

The first example uses the *log* keyword to record a log message every time the ACL makes a match. Here are some log messages generated by this command:

```
Feb  6 13:01:19: %SEC-6-IPACCESSLOGRP: list 150 permitted ospf 10.1.1.1 ->
  224.0.0.5, 9 packets
Feb  6 13:01:19: %SEC-6-IPACCESSLOGDP: list 150 permitted icmp 10.1.1.1 ->
  10.1.1.2 (0/0), 4 packets
```

You can also get a breakdown of how many matches each line in the ACL has recorded with the *show*

access-list command:

```
Router1#show access-list 150
Extended IP access list 150
    permit ip any any log (15 matches)
Router1#
```

The second form, with the *log-input* keyword, causes the router to include other useful data in the log messages. With this option, the log messages will include the port where the packet was received:

```
Feb  6 13:08:31: %SEC-6-IPACCESSLOGP: list 150 permitted tcp 10.1.1.1(0)
  (Serial0/1 ) -> 10.1.1.2(0), 80 packets
Feb  6 13:08:38: %SEC-6-IPACCESSLOGP: list 150 permitted tcp 10.2.2.2(0)
  (Serial0/1 ) -> 172.25.26.5(0), 1 packet
Feb  6 13:10:29: %SEC-6-IPACCESSLOGP: list 150 permitted tcp 10.2.2.2(0)
  (Serial0/1 ) -> 172.20.100.1(0), 1 packet
```

If we apply this ACL on an Ethernet or Token Ring port, the log messages will also include MAC address information:

```
Feb  6 14:56:34: %SEC-6-IPACCESSLOGP: list 150 permitted tcp 172.25.1.1(0)
(FastEthernet0/0.1 0010.4b09.5700) -> 172.25.25.1(0), 1 packet
Router1#
Feb  6 14:58:20: %SEC-6-IPACCESSLOGP: list 150 permitted tcp 172.25.1.7(0)
(FastEthernet0/0.1 0000.0c92.bc6a) -> 172.25.1.5(0), 1 packet
```

The only problem with these commands is that they tend to produce huge numbers of log messages. To be really useful, we recommend using this feature in conjunction with a remote log server, as described in Chapter 18. Then you can store and analyze all of the messages without worrying that you will lose information when the router's internal log buffer overwrites itself. We offer a useful script for analyzing the messages and look for important patterns in Recipe 19.10.

In general, we recommend logging all denied packets, because they tend to represent the rejected traffic that is not part of the normal functioning of the network.

While all of the examples in this recipe used extended ACLs, the *log* keyword is also available with standard ACLs:

```
Router1(config)#access-list 77 permit any log
```

However, the *log-input* option is available only for extended ACLs.

19.8.4 See Also

Recipe 19.10

Top

Recipe 19.9 Logging TCP Sessions

19.9.1 Problem

You want to log the total number of TCP sessions.

19.9.2 Solution

You can configure the router to log the total number of TCP sessions, rather than just the number of packets, with the following set of commands:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#access-list 122 permit tcp any any eq telnet established
Router1(config)#access-list 122 permit tcp any any eq telnet
Router1(config)#access-list 122 permit ip any any
Router1(config)#interface Serial0/0
Router1(config-if)#ip access-group 122 in
Router1(config-if)#end
Router1#
```

Here is an alternative method that will also work:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#access-list 121 permit tcp any any eq telnet syn
Router1(config)#access-list 121 permit tcp any any eq telnet
Router1(config)#access-list 121 permit ip any any
Router1(config)#interface Serial0/0
Router1(config-if)#ip access-group 121 in
Router1(config-if)#end
Router1#
```

19.9.3 Discussion

When you configure an access list, the router counts the total number of times it finds something that matches each line in the ACL. While this information is often useful, it does not tell you whether these counters are recording a thousand packets on a single session, or a single packet from each of a thousand sessions. The ACLs in this recipe count the number of TCP sessions, as well as the total number of packets.

In the first example, the first line in the ACL permits all established Telnet packets to pass through the access list, as we did in Recipe 19.5. The second line then matches all of the Telnet packets that the first one does not, which mainly means the initial SYN packet that starts the TCP session. As we mentioned in Recipe 19.4, the first packet of a TCP session contains the SYN bit. And, as we discussed in Recipe 19.5, an ACL that includes the *established* keyword will not match any packets that have the SYN bit set.

So, the second line catches the initial session establishment, while the first line matches all of the other packets in the session. Therefore, the second line will give us a way to count the total number of TCP sessions that pass through the router. Note that these sessions can be between any two devices—as long as they communicate through this router, we can count them. Of course, the ACL in the example counts only Telnet sessions, that is, sessions on TCP port number 23. However, it is easy enough to change the port number in the ACL to monitor other TCP-based applications.

When you apply this ACL to an interface, the *show access-list* command shows a running count of the number of Telnet sessions that have occurred:

```
Router1#show access-list 122
Extended IP access list 122
     permit tcp any any eq telnet established (3843 matches)
    permit tcp any any eq telnet (6 matches)
    permit ip any any (31937 matches)
Router1#
```

Six separate Telnet sessions have passed through the interface where we applied this ACL. If you want to know the total number of Telnet packets, you can simply add the first two lines together: 3843+6=3849 packets.

The second example uses a slightly different method for counting the number of sessions. In this case, the first line of the access list matches only Telnet packets with the SYN bit set, as discussed in Recipe 19.4:

```
Router1(config)#access-list 121 permit tcp any any eq telnet syn
```

The only packets that have this bit set are the packets from the initial TCP three-phase handshake that establishes the session. So this also gives us a way of counting the total number of Telnet sessions. The second line of this ACL captures the remaining Telnet packets:

```
Router1#show access-list 121
Extended IP access list 121
    permit tcp any any eq telnet syn (7 matches)
    permit tcp any any eq telnet (3057 matches)
    permit ip any any (9404 matches)
Router1#
```

This ACL has counted seven separate Telnet sessions: 7+3057=3064 total Telnet packets.

We can take the counting functionality of these ACLs a step further by adding the *log* keyword to the ACL lines that count the sessions:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#access-list 121 permit tcp any any eq telnet syn log
Router1(config)#access-list 121 permit tcp any any eq telnet
Router1(config)#access-list 121 permit ip any any
Router1(config)#end
Router1#
```

Including the *log* keyword like this allows us to keep a log of every TCP session, without needing to log all of the packets in these sessions. This can be useful for security records and audits:

```
Router1#show logging | include list 121
Feb  7 15:36:13: %SEC-6-IPACCESSLOGP: list 121 permitted tcp 172.25.1.1(3886)
-> 10.2.2.2(23), 1 packet
Feb  7 15:36:39: %SEC-6-IPACCESSLOGP: list 121 permitted tcp 172.25.1.1(3887)
-> 10.2.2.2(23), 1 packet
Feb  7 15:38:32: %SEC-6-IPACCESSLOGP: list 121 permitted tcp 172.25.1.1(3888)
-> 10.2.2.2(23), 1 packet
Feb  8 07:48:20: %SEC-6-IPACCESSLOGP: list 121 permitted tcp 172.25.1.1(4332)
 -> 10.2.2.2(23), 1 packet
Feb  8 07:49:35: %SEC-6-IPACCESSLOGP: list 121 permitted tcp 172.25.1.1(4333)
-> 10.2.2.2(23), 1 packet
Feb  8 08:08:57: %SEC-6-IPACCESSLOGP: list 121 permitted tcp 172.25.1.1(4339)
-> 10.2.2.2(23), 1 packet
Router1#
```

For more information about logging, see Chapter 18.

19.9.4 See Also

Recipe 19.4; Recipe 19.5; Chapter 18

Top

Recipe 19.10 Analyzing ACL Log Entries

19.10.1 Problem

You want to analyze the log entries created by logging ACLs.

19.10.2 Solution

The Perl script in Example 19-1 parses a router syslog file and builds a detailed report of packets that were denied by logging ACLs. By default, the script will parse every ACL log message that it finds in the syslog file on a server. You can also look for messages associated with a particular ACL by specifying the ACL number or name as a command-line argument.

Example 19-1. logscan.pl

```
#!/usr/local/bin/perl
#
#       logscan.pl -- a script to extract ACL logs from a syslog file.
#
# Set behaviour
$log="/var/log/cisco.log";
$ntop=10;
#
chomp  ($acl=$ARGV[0]);
if ($acl =  = "") { $acl=".*"};

open(LOG , "<$log") or die;
while (<LOG>) {
 if (/IPACCESSLOGP: list $acl denied ([tcpud]+) ([0-9.]+)\(([0-9]+)\) ->
  ([0-9.]+)\(([0-9]+\), ([0-9]+) /) {
    $x=$6;
    $srca{$2}+=$x;
    $foo=sprintf("%16s  -> %16s  %3s port %-6s",$2,$4,$1,$5);
    $moo=sprintf("%3s port %-6s",$1,$5);
   $quad{$foo}+=$x;
   $port{$moo}+=$x;
 }
}
$n=0;
printf ("Connection Summary:\n");
foreach $i (sort { $quad{$b} <=> $quad{$a} } keys %quad) {
   if ($n++ >= $ntop) { last };
```

```
    printf ("%6s:%s\n", $quad{$i},$i);
}
$n=0;
printf ("\nDestination Port Summary:\n");
foreach $i ( sort { $port{$b} <=> $port{$a} } keys %port) {
   if ($n++ >= $ntop) { last };
    printf ("%6s: %s\n", $port{$i},$i);
}
$n=0;
printf ("\nSource Address Summary:\n");
foreach $i ( sort { $srca{$b} <=> $srca{$a} } keys %srca) {
   if ($n++ >= $ntop) { last };
    printf ("%6s: %s\n", $srca{$i},$i);
}
```

Note that we have had to split the line that begins "if (/IPACCESSLOGP: list" across two lines so that it will fit on the page. If you decide to use this script, please type this as a single line.

19.10.3 Discussion

It's a good idea configure your access lists so that they log all of the packets that they deny. This is particularly true for security-related ACLs. You can then use these log messages for security or audit purposes. Unfortunately, this level of logging can create a large number of messages, which makes analysis difficult. This Perl script automates the most difficult part of this analysis by parsing a large file of ACL log messages and building a report that can help to identify potential problems.

The report produced by the script has three sections that summarize connections, destination ports, and source IP addresses. The connection report displays the top 10 most common connection denied attempts including the source address, destination source, and destination port number. The destination port summary displays the top 10 most frequently denied destination ports. Finally, the source address summary displays the 10 hosts whose connection attempts were most frequently denied. In each case, the script looks at both TCP and UDP ports.

The following is a sample report:

```
Freebsd%./logscan.pl
Connection Summary:
   195:      172.25.1.1  ->     172.20.100.1  tcp port 23
    13:      172.25.1.1  ->     172.20.100.1  tcp port 22
     8:      172.20.1.2  ->       172.25.1.3  udp port 53
     6:      172.20.1.2  ->       172.25.1.1  udp port 123
     6:      172.25.1.1  ->         10.2.2.2  tcp port 23
     4:      172.20.1.2  ->       172.25.1.1  udp port 162
     4:      172.25.1.1  ->     172.20.100.1  tcp port 21
     4:      172.20.1.2  ->       172.25.1.1  udp port 53
     3:      172.25.1.1  ->     172.20.100.1  tcp port 80
     2:      172.20.1.2  ->       172.25.1.3  udp port 123
```

```
Destination Port Summary:
   206: tcp port 23
    14: tcp port 22
    12: udp port 53
     8: udp port 123
     4: tcp port 80
     4: udp port 162
     4: tcp port 21
     2: tcp port 443
     1: tcp port 6000
     1: tcp port 79

Source Address Summary:
   222: 172.25.1.1
    24: 172.20.1.2
     3: 172.25.1.6
     1: 192.168.4.217
     1: 172.25.1.9
     1: 10.2.3.134
     1: 172.25.1.4
     1: 172.22.1.9
     1: 10.23.55.121
     1: 172.25.1.3
Freebsd%
```

This report can help identify troubling behavior that warrants further investigation. For instance, address `172.25.1.1` has attempted to telnet to `172.20.100.1` 195 times. This may not have been evident by scanning the log file by hand.

The script can also build a report based on the messages received from a particular ACL number or name:

```
Freebsd%./logscan.pl 122
Connection Summary:
     2:      172.25.1.6  ->         10.2.2.2  tcp port 443
     1:   192.168.4.217  ->         10.2.2.2  tcp port 23
     1:      172.22.1.9  ->         10.2.2.2  tcp port 6000
     1:      10.2.3.134  ->         10.2.2.2  tcp port 23
     1:      172.25.1.9  ->         10.2.2.2  tcp port 23
     1:    10.23.55.121  ->         10.2.2.2  tcp port 79
     1:      172.25.1.1  ->         10.2.2.2  tcp port 22
     1:      172.25.1.6  ->         10.2.2.2  tcp port 23

Destination Port Summary:
     4: tcp port 23
     2: tcp port 443
     1: tcp port 22
     1: tcp port 6000
     1: tcp port 79
```

```
Source Address Summary:
     3: 172.25.1.6
     1: 10.2.3.134
     1: 172.22.1.9
     1: 192.168.4.217
     1: 172.25.1.1
     1: 172.25.1.9
     1: 10.23.55.121
Freebsd%
```

Before you can use this script you must modify the variable *$log*. This variable must contain the full directory and filename of the syslog file that you wish to scan. The script is then ready to launch. The only other variable you may want to modify is the *$ntop* variable. This variable defines how long each section of the report will be. By default, the script is set to display the top 10 matches in each category. However, you can set this number to any number you prefer. Note also that, as in the previous example, if there are fewer than *$ntop* matches in any category, the script will show only the matches that it actually finds.

For more information on logging and remote logging to a syslog server, please see Chapter 18.

19.10.4 See Also

Recipe 19.8; Recipe 19.9; Chapter 18

Top

Recipe 19.11 Using Named and Reflexive Access Lists

19.11.1 Problem

You want to use a reflexive ACL, embedded in a named ACL.

19.11.2 Solution

A basic named ACL is similar to the numbered ACLs that we discussed earlier in this chapter. They can work like either standard or extended IP ACLs:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#ip access-list standard STANDARD-ACL
Router1(config-std-nacl)#remark This is a standard ACL
Router1(config-std-nacl)#permit any log
Router1(config-std-nacl)#exit
Router1(config)#ip access-list extended EXTENDED-ACL
Router1(config-ext-nacl)#remark This is an extended ACL
Router1(config-ext-nacl)#deny tcp any any eq www
Router1(config-ext-nacl)#permit ip any any log
Router1(config-ext-nacl)#exit
Router1(config)#interface Serial0/1
Router1(config-if)#ip access-group STANDARD-ACL in
Router1(config-if)#end
Router1#
```

You can embed a reflexive ACL inside of a named extended IP ACL. The *reflect* keyword defines the reflexive ACL rule, and the *evaluate* command executes it. The following example filters ICMP packets so that you can initiate a *ping* test from one side of the network, but not the other:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#ip access-list extended PING-OUT
Router1(config-ext-nacl)#permit icmp any any reflect ICMP-REFLECT timeout 15
Router1(config-ext-nacl)#permit ip any any
Router1(config-ext-nacl)#exit
Router1(config)#ip access-list extended PING-IN
Router1(config-ext-nacl)#evaluate ICMP-REFLECT
Router1(config-ext-nacl)#deny icmp any any log
Router1(config-ext-nacl)#permit ip any any
Router1(config-ext-nacl)#exit
Router1(config)#interface Serial0/1
```

```
Router1(config-if)#ip access-group PING-OUT out
Router1(config-if)#ip access-group PING-IN in
Router1(config-if)#end
Router1#
```

19.11.3 Discussion

The first example in this recipe demonstrates how to use named ACLs. There is very little difference between this example and the one shown in Recipe 19.1, except that here we have used a different type of ACL to accomplish the same thing. One useful difference between the two versions is that you can delete an individual rule from a named ACL:

```
Router1#show access-list EXTENDED-ACL
Extended IP access list EXTENDED-ACL
    deny tcp any any eq www
    permit ip any any log
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#ip access-list extended EXTENDED-ACL
Router1(config-ext-nacl)#no deny tcp any any eq www
Router1(config-ext-nacl)#end
Router1#show access-list EXTENDED-ACL
Extended IP access list EXTENDED-ACL
    permit ip any any log
Router1#
```

However, as with numbered ACLs, you cannot add individual rules to the middle of a named ACL.

Named ACLs start to show their real value, though, when you need to use more advanced features such as reflexive ACLs, as we did in the second example. This example is similar in spirit to what we did to restrict TCP sessions in Recipe 19.5. In that case, we wanted to ensure that users on the trusted side of the network could initiate TCP connections to the untrusted side, but reject any incoming connection attempts. Here we do the same thing with ICMP packets.

Of course, because TCP is a connection-oriented protocol, it is not quite as difficult to determine which side initiated the session. But because ICMP doesn't have the concept of a session, we have to wait until somebody on the inside sends an ICMP packet to somebody on the outside. When this happens, we tell the router that it can expect to see an appropriate ICMP response from the same IP address, so it should let that packet through.

Let's look at the outbound ACL first:

```
Router1(config)#ip access-list extended PING-OUT
Router1(config-ext-nacl)#permit icmp any any reflect ICMP-REFLECT timeout 15
Router1(config-ext-nacl)#permit ip any any
```

The first *permit* command includes the keyword *reflect*, and defines the reflection rule name as *ICMP-REFLECT*. We have applied this ACL to watch for outbound packets on the interface. As soon as we

send out an ICMP packet, such as a *ping* query, the router starts looking for the reflected version of this packet—in this case, a *ping* response.

We have also included the *timeout* keyword at the end of the line with an argument of *15*. This tells the router that it should not wait more than 15 seconds after the last outbound packet for additional inbound packets.

The inbound rule then uses the *evaluate* keyword to dynamically enable the reflection rule:

```
Router1(config)#ip access-list extended PING-IN
Router1(config-ext-nacl)#evaluate ICMP-REFLECT
Router1(config-ext-nacl)#deny icmp any any log
Router1(config-ext-nacl)#permit ip any any
```

Note that this example uses the same rule name, *ICMP-REFLECT*, that was previously defined in the outbound ACL. If the incoming packet looks like a reflected version of whatever was defined when we created this rule, the ACL will permit the packet. If the packet doesn't match this rule, then it will continue checking the rest of the ACL normally. In this case, we have followed the *evaluate* command with a command that will explicitly deny all other ICMP packets that don't match the reflection rule.

Note that the router will check the reflected packet to ensure that it has the correct source and destination addresses, based on the outbound packet. For example, if you use reflexive ACLs to match a UDP application, the router will also check port numbers to ensure that the inbound packet is legitimate.

19.11.4 See Also

Recipe 19.1; Recipe 19.5

Top

Recipe 19.12 Dealing with Passive Mode FTP

19.12.1 Problem

You want to construct an ACL that can identify passive mode FTP sessions.

19.12.2 Solution

This example shows how to filter passive FTP control and data sessions:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#access-list 144 permit tcp any gt 1023 any eq ftp
Router1(config)#access-list 144 permit tcp any gt 1023 any gt 1023
Router1(config)#access-list 144 deny ip any any
Router1(config)#interface Serial0/0.1
Router1(config-subif)#ip access-group 144 in
Router1(config-subif)#end
Router1#
```

19.12.3 Discussion

In Recipe 19.6, we briefly reviewed the traditional way that FTP works. However, there is another subtle variation on this process, which is commonly called passive FTP. The user connects the control session to the server on port 21, exactly as before. But in the passive FTP case, the client software issues the command PASV. This command instructs the server to listen on a new non-default data port and wait for a connection. The server selects a new port, which it tells to the client. The server then opens this port and waits for a connection. The client device initiates a new TCP connection to this temporary port number and uses that connection to transfer its data.

This may sound like an unusual way of doing things, and it probably is. However, it is actually the default mode for many web browsers that perform FTP file transfers, including Internet Explorer and Netscape. This makes passive FTP the most common FTP mode for many networks. The problem is that if you want to control this traffic using an ACL of any kind, you no longer know the source or destination TCP port numbers. For example, if you need to restrict some traffic while ensuring that passive FTP is allowed, you will need an ACL that can somehow permit the temporary port numbers. In Recipe 19.13, we demonstrate a filtering method in which the router learns about the new port by watching the control session of the FTP session.

This example takes a simpler approach and uses an extended ACL to deal with passive FTP. The trouble with this ACL is that it opens all TCP ports from 1024 and above. Clearly, this is not desirable on a router facing the Internet or some other potentially unfriendly network.

Although our example permits passive FTP to pass through, it opens up over 64,000 TCP ports in the process. Obviously, this is not the best way to permit passive FTP. In Recipe 19.13, we discuss a much more secure method of allowing passive FTP through your router.

19.12.4 See Also

Recipe 19.6; Recipe 19.13

Top

Recipe 19.13 Using Context-Based Access Lists

19.13.1 Problem

You want to use your router as a firewall to perform advanced filtering functionality.

19.13.2 Solution

The following example shows how to configure the router to perform *stateful* inspection of TCP or UDP packets:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#access-list 166 deny   ip any any
Router1(config)#access-list 167 permit tcp any any eq telnet
Router1(config)#ip inspect name Telnet tcp
Router1(config)#interface Serial0/1
Router1(config-if)#ip access-group 166 in
Router1(config-if)#ip access-group 167 out
Router1(config-if)#ip inspect Telnet out
Router1(config-if)#end
Router1#
```

> Cisco's firewall IOS feature set must be installed on a router before you can configure context-based access lists.

19.13.3 Discussion

CBAC has been available as part of the IOS firewall feature set since Version 11.2(P). CBAC does a stateful inspection of TCP and UDP packets to manage sessions as they pass through the router. It uses this state information to dynamically modify existing extended ACLs to control the active sessions. CBAC can also monitor and manage sessions based on application type and identify, terminate, or log any suspicious activity.

CBAC provides much greater security than a regular filtering ACL because it uses features similar to those found in dedicated firewalls. In fact, the IOS firewall feature set (including CBAC) makes an excellent firewall for small or cost-conscious organizations. Although it is not suitable for every application, CBAC's stateful inspection does provide better security than simple packet filtering

firewalls.

CBAC works by inspecting active sessions and dynamically creating temporary ACL entries to allow the return traffic through. In our example, we configured the router with an inbound ACL that denies all IP packets, which would normally prevent the router from accepting any inbound IP traffic on this interface. However, when somebody on the inside of the network initiates an outbound connection through this port, CBAC creates a temporary ACL that allows the device on the outside to respond.

We can demonstrate this by initiating an outbound Telnet session through this interface, then viewing the inbound ACL:

```
Router1#show ip access-list 166
Extended IP access list 166
    permit tcp host 10.2.2.2 eq telnet host 172.25.1.1 eq 1379 (22 matches)
    deny ip any any (456 matches)
Router1#
```

Note that the original ACL now contains a new entry to allow the return data of the Telnet session that we originated. This temporary ACL entry includes the exact source and destination IP addresses and port numbers. This ensures that our Telnet session works normally, but it prevents all other possible IP addresses from connecting to the client device's source port. In fact, CBAC even prevents the external server from connecting to the internal device's source port using a new session because it maintains specific session information such as source and destination port numbers and TCP sequence numbers. This temporary ACL entry will terminate when the session ends, leaving only the static *deny all* entry that we originally configured in ACL number 166.

This configuration will block all inbound connection attempts. A common and very simple technique used by hackers as a prelude to attacking an Internet site is to perform a port scan. This means systematically trying to start an inbound session on each port in a large range. If there is any response at all, the attacker knows that there is something listening, which presents a useful starting point for breaking in. However, because this CBAC configuration blocks all unsolicited connection attempts regardless of the port, port scanning reveals nothing useful. This is usually enough to deter all but the most sophisticated hackers.

In fact, the only effective way to get a packet past the ACL configuration shown in this recipe is to send a packet with the right source and destination addresses and port numbers as well as the right TCP sequence number. This technique is called a *hijack* attack, and it is almost impossible to prevent. Attackers using this technique are generally able to get a single packet past the firewall. In some cases an attacker can use this single packet to cause mischief. Fortunately for us, hijack attacks are extremely difficult to execute.

The *show ip inspect sessions* command lets you view information about all of the sessions that CBAC is currently watching:

```
Router1#show ip inspect sessions
```

```
Established Sessions
 Session 821061C0 (172.25.1.1:1379)=>(10.2.2.2:23) tcp SIS_OPEN
Router1#
```

The example shows how to inspect generic TCP sessions. However, one of CBAC's greatest strengths is its ability to identify application-specific behavior and adjust its ACL entries accordingly. Table 19-2Table 19-2 displays the various applications that CBAC is able to monitor.

Table 19-3. CBAC application support keywords

Keyword	Application protocol
cuseeme	CU-SeeMe protocol
fragment	IP fragmented packets
ftp	File Transfer Protocol
h323	H.323 protocol
http	HTTP (Java blocking)
netshow	Microsoft's NetShow
rcmd	Unix R-commands
realaudio	RealAudio
rpc	Sun RPC
rtsp	Microsoft's RPC
smtp	Simple Mail Transfer Protocol
sqlnet	SQL*Net
streamworks	StreamWorks
tcp	Generic TCP
tftp	Trivial File Transfer Protocol
udp	Generic UDP
vdolive	VDOLive

Just because an application is not listed in this table doesn't mean that CBAC will not manage its sessions. In fact, the Telnet example in this recipe used generic TCP inspection. Unless your application does something unusual in its session setup and negotiation sequences, the generic

inspection should work well. For example, standard HTTP also uses standard TCP session rules. The special HTTP option for CBAC simply allows it to handle Java.

Let's take a look at an application that doesn't use standard TCP ports to build connections: passive FTP. As we indicated in Recipe 19.12, passive FTP is difficult to secure because the server can choose to use any of a large number of ports for its data session. The result is that both the source and destination ports are determined dynamically when the session is established. Recipe 19.12 showed a way to filter this traffic statically, but with the unfortunate side effect of leaving some 64,000 other ports open. With CBAC, this is not necessary—the router can watch the FTP control session and determine the server's data port. This allows it to open only the required port:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#access-list 155 permit tcp any any eq ftp
Router1(config)#access-list 155 deny ip any any
Router1(config)#ip inspect name TEST ftp
Router1(config)#interface Serial0/0
Router1(config-subif)#ip access-group 155 in
Router1(config-subif)#ip inspect TEST in
Router1(config-subif)#end
Router1#
```

In this example, the *ftp* keyword invokes FTP inspection instead of the generic TCP inspection that we showed in the previous example. Also, notice that the first line of our ACL includes the ftp keyword, which permits only the FTP control session. This is necessary so that CBAC will permit the control part of the application to pass.

Now we can connect to the FTP server using a web browser. The ACL now shows the new entries that CBAC has created for us:

```
Router1#show ip access-list 155
Extended IP access list 155
    permit tcp host 172.20.1.2 eq 11252 host 172.25.1.3 eq 49155 (1415 matches)
    permit tcp any any eq ftp (151 matches)
    deny ip any any (3829 matches)
Router1#
```

CBAC monitored the FTP control session and opened the specific FTP data ports for the passive FTP data connection. You can tell that this is a passive FTP session from the high port numbers.

CBAC inspection can also handle normal FTP. CBAC FTP inspection blocks third party connections, allowing only "safe" FTP data ports (1024-65535). CBAC monitors the FTP control session, and will not open a data port if the client authentication fails.

Although we haven't shown an example for UDP, CBAC can also handle UDP-based applications in a similar fashion to TCP. When you send a UDP packet out through the interface, CBAC knows to expect an appropriate response. UDP is difficult to handle because it is not session oriented, but CBAC

does it well.

CBAC also offers application support for TFTP, which is a feature rarely seen in commercial firewalls. The TFTP server uses the well-known UDP port 69. The client sends an initial packet to the server on this well-known port. But then the server opens a new arbitrary port greater than 1023 for the duration of the TFTP session. A standard packet filter would have to permit all UDP ports above 1023 to let TFTP work. CBAC gets around this problem by monitoring the TFTP session to determine which UDP port to open.

CBAC has a several settings that you can adjust to improve the overall performance and security of the router. We've listed some of the most important options in Table 19-3Table 19-3.

Table 19-4. Recommended CBAC settings

Setting	Description	Default	Recommended
TCP idle-time	Length of time CBAC will continue to permit an idle TCP session	1 hour	30 minutes
UDP idle-time	Length of time CBAC will continue to permit an idle UDP session	30 seconds	20 seconds
Finwait-time	Length of time CBAC will continue to permit a TCP session after the exchange of FIN packets	5 seconds	1 second
Synwait-time	Length of time that CBAC will wait after receiving a SYN packet without completing the session establishment	30 seconds	15 seconds

The following set of commands configures the timer settings of CBAC shown in Table 19-3:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#ip inspect tcp idle-time 1800
Router1(config)#ip inspect udp idle-time 20
Router1(config)#ip inspect tcp finwait-time 1
Router1(config)#ip inspect tcp synwait-time 15
Router1(config)#end
Router1#
```

You can view the CBAC configuration settings with the *show ip inspect config* command:

```
Router1#show ip inspect config
Session audit trail is disabled
Session alert is enabled
one-minute (sampling period) thresholds are [400:500] connections
max-incomplete sessions thresholds are [400:500]
```

```
max-incomplete tcp connections per host is 50. Block-time 0 minute.
tcp synwait-time is 15 sec -- tcp finwait-time is 1 sec
tcp idle-time is 1800 sec -- udp idle-time is 20 sec
dns-timeout is 5 sec
Inspection Rule Configuration
  Inspection name Telnet
    tcp alert is on audit-trail is off timeout 1800

Router1#
```

CBAC also provides a method of logging managed sessions by enabling audit trails. You can enable an audit trail on a particular CBAC inspection command with the *audit-trail* keyword:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#ip inspect name Telnet tcp audit-trail on
Router1(config)#end
Router1#
```

When you enable CBAC audit trails, the router will create a log entry after each session terminates. This log entry includes detailed information about the connection. Here a sample audit trail log that the router recorded after we terminated a Telnet session:

```
Feb  8 14:37:24: %FW-6-SESS_AUDIT_TRAIL: tcp session initiator
(172.25.1.1:1402) sent 59 bytes -- responder (10.2.2.2:23) sent 1299
```

For more information on logging please see Chapter 18.

19.13.4 See Also

Recipe 19.12; Chapter 18.

Top

Chapter 20. DHCP

Top

Introduction

Dynamic Host Configuration Protocol (DHCP) is often used on networks to allow end devices to automatically retrieve their network configuration when they first connect to the network. It basically expands on the earlier Bootstrap Protocol (BOOTP) and uses the same UDP ports, numbers 67 and 68. The protocol itself is defined in RFC 2131, and the configuration options are defined in RFC 2132.

The most common application for DHCP is to automatically set up IP addresses, netmasks, and default gateways for end devices. However, the protocol can also configure many other options, such as DNS servers, domain names, time zones, NTP servers, and many others. Some software vendors have even added their own configuration options to automatically set up key applications on end devices.

DHCP makes it possible to give a minimal common configuration to all user workstations. You can simply plug the device into the network at any point, and DHCP will take care of getting an IP address that will work at that location. This minimizes errors due to manual configuration, centralizes control over configuration information, and greatly reduces technician costs because anybody can connect a device to the network.

There are three distinct element types in a DHCP network. There must be a client and a server. If these two elements are not on the same Layer 2 network, there also must be a proxy, which usually runs on the router. The proxy is needed because the client device initially doesn't know its own IP address, so it must send out a Layer 2 broadcast to find a server that has this information. The router must relay these broadcasts to the DHCP server, then forward the responses back to the correct Layer 2 address so that the right end device gets the right configuration information.

Historically, the router's only role in BOOTP or DHCP was this proxy function. However, Cisco has recently added both DHCP client and server functionality. This chapter will show configuration examples for all three of these functions, but many of the recipes will focus on complex server configurations.

A DHCP exchange starts with a client device, such as an end user workstation. Typically, this device will boot and connect to the network with no preconfigured network information. It doesn't know its IP address, the address of its router, its subnet, or netmask. It doesn't even know the address of the server that will provide these pieces of information. The only thing it can do is send out a UDP broadcast packet looking for a server.

Most DHCP networks of any size include two or more DHCP servers for redundancy. The end devices typically just need to talk to a DHCP server at startup time, but they will not work at all without it. So redundancy is important. This also means that it is not unusual for an end device to see several responses to a DHCP request. It will generally just use the first response. However, this also

underscores the importance of ensuring that all of the DHCP servers distribute the same information. Their databases of end device configuration parameters must be synchronized.

The end device requests configuration information from one of the servers. It must specify exactly what options it requires. The server does not need to respond to all of the requested options, but it cannot offer additional unrequested information to the client, even if it has additional information in its database. This is an important detail to remember—it can be very confusing when an end device has some manually configured options that are not replaced by the information on the server.

Since duplicate IP addresses can cause serious problems on a network, most DHCP servers track address conflicts. They do this by attempting to *ping* each IP address before telling an end device that it is safe to use the address. Many DHCP clients will also double-check that the address is not already in use by sending an ARP request before using it. However, neither of these checks is mandatory, and some DHCP clients and servers do not check before using an address.

One of the important features of DHCP is the ability to allocate IP addresses for a configurable period of time, called the *lease* period. If a client device wants to keep its IP address for longer than this period, it must renew the lease before it expires. Clients are free to renew their leases as often as they like.

The server can allocate IP addresses from a pool on a first-come, first-served basis, or it can associate IP addresses with end device MAC addresses to ensure that a particular client always receives the same address.

Top

Recipe 20.1 Using IP Helper Addresses for DHCP

20.1.1 Problem

You want to configure your router to pass DHCP requests from local clients to a centralized DHCP server.

20.1.2 Solution

The *ip helper-address* configuration command allows the router to forward local DHCP requests to one or more centralized DHCP servers:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#interface Ethernet0
Router1(config-if)#ip helper-address 172.25.1.1
Router1(config-if)#ip helper-address 172.25.10.7
Router1(config-if)#end
Router1#
```

20.1.3 Discussion

The traditional role of a router in DHCP has been to act as a proxy device that forwards information between the client and server. This proxy function is still the most common for routers, but Cisco routers have had DHCP server and client features since IOS level 12.0(1)T.

Because the initial DHCP request comes from a client that typically doesn't have an IP address, it must find the server using a Layer 2 broadcast. So, if the router is not able to function as a proxy for these broadcasts, there must be a DHCP server on every network segment.

The DHCP server needs two critical pieces of information before it can allocate an IP address to the client: the subnet that the client is connected to and the client device's MAC address. The subnet information is needed to ensure that the address that the server allocates will actually work on client's network segment, while the MAC address allows the server to find any information that is unique to a workstation. This is essential if you need to ensure that a particular end device gets the same IP address every time it connects to the network.

So the DHCP proxy, which is the router itself, must convert the local broadcast from the client to a unicast packet and forward it to the server. This is what the *ip helper-address* command does.

When the DHCP client sends the DHCP request packet, it doesn't have an IP address. So it uses the all-zeros address (`0.0.0.0`) as the IP source address. It also doesn't know how to reach the DHCP server, so it uses a general broadcast address (`255.255.255.255`) for the destination.

The router must replace the source address with its own IP address, for the interface that received the request. And it replaces the destination address with the address specified in the *ip helper-address* command. The client device's MAC address is included in the payload of the original DHCP request packet, so the router doesn't need to do anything to ensure that the server receives this information.

The DHCP server now has enough information to assign an address from the correct address pool, since it now knows what the originating subnet was for the DHCP request. The server then sends a unicast response back to the proxy router, which in turn sends the request back to the correct MAC address.

The example shows two *ip helper-address* commands. You should include one of these commands for each of your DHCP servers. The router will forward the DHCP broadcasts to all of these addresses. Most organizations use at least two DHCP servers because, although the utilization is light, the functionality is critical. If the client device receives several responses to a DHCP request, it will usually select the one it receives first.

It is important to note that the *ip helper-address* command does not just forward DHCP requests. In fact, although you can configure it to forward any UDP broadcast you want, it will forward UDP broadcast packets for several different UDP ports to the specified address by default. In some cases, this unwanted traffic can cause problems on the network or the DHCP server. Recipe 20.2 focuses on this issue.

The *show ip interface* command includes information about the helper addresses configured on an interface:

```
Router1#show ip interface Ethernet0
Ethernet0 is up, line protocol is up
  Internet address is 192.168.30.1/24
  Broadcast address is 255.255.255.255
  Address determined by setup command
  MTU is 1500 bytes
  Helper addresses are 172.25.1.3
                       172.25.1.1
  Directed broadcast forwarding is disabled
  <lines removed for brevity>
Router1#
```

20.1.4 See Also

Recipe 20.2

Top

Recipe 20.2 Limiting the Impact of IP Helper Addresses

20.2.1 Problem

After configuring your router to use IP helper addresses, you suffer from high link utilization or high CPU utilization on the DHCP server.

20.2.2 Solution

The *ip helper-address* command implicitly enables forwarding of several different kinds of UDP broadcasts. You can prevent the router from forwarding the unwanted types of broadcasts with the *no ip forward-protocol udp* configuration command:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#no ip forward-protocol udp tftp
Router1(config)#no ip forward-protocol udp nameserver
Router1(config)#no ip forward-protocol udp domain
Router1(config)#no ip forward-protocol udp time
Router1(config)#no ip forward-protocol udp netbios-ns
Router1(config)#no ip forward-protocol udp netbios-dgm
Router1(config)#no ip forward-protocol udp tacacs
Router1(config)#end
Router1#
```

20.2.3 Discussion

As mentioned in Recipe 20.1, if the DHCP client and server are on different network segments, the router on the client's segment must be configured with a helper address for DHCP to work. However, the helper address configuration forwards a variety of different UDP broadcasts, not just DHCP packets. This can cause network and CPU loading problems on the DHCP server.

By default, when you configure the *ip helper-address* command on an interface, the router will automatically forward UDP broadcasts for all of the protocols shown in Table 20-1.

Table 20-1. Default UDP protocols for helper addresses

Type	Description	UDP port
bootpc	Bootstrap or DHCP client	68
bootps	Bootstrap or DHCP server	67
domain	Domain Name Service (DNS)	53
nameserver	IEN-116 name service (obsolete)	42
netbios-dgm	NetBIOS datagram service	138
netbios-ns	NetBIOS name service	137
tacacs	TAC Access Control System	49
time	Time	37
tftp	Trivial File Transfer Protocol	69

In particular, networks that include Microsoft Windows networking features use a lot of NetBIOS packets. The DHCP server receives broadcasts from many end device segments throughout the network. It is possible to have enough traffic aggregating on this point to cause serious problems.

This recipe disables each unnecessary protocol, one at a time, using the *no ip forward-protocol* configuration command. Some organizations choose to disable only the NetBIOS protocol because this is usually the one that most frequently causes problems.

We strongly recommend using the *no ip forward-protocol* command to ensure that only the required protocols are being forwarded to your DHCP server. Note, however, that this command cannot forward different protocols to different helper addresses. If you have two different servers handling different UDP broadcast protocols, they will both receive all of the local broadcasts that the router accepts. If you need more detailed control over these types of applications, you may find that the broadcast-to-multicast conversion features discussed in Chapter 23 are more effective.

20.2.4 See Also

Recipe 20.1; Chapter 23

Top

Recipe 20.3 Using DHCP to Dynamically Configure Router IP Addresses

20.3.1 Problem

You want the router to dynamically obtain its IP addressing information.

20.3.2 Solution

The *ip address dhcp* configuration command allows the router to dynamically obtain the address information for an interface:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#interface Ethernet0
Router1(config-if)#ip address dhcp client-id Ethernet0
Router1(config-if)#end
Router1#
Interface Ethernet0 assigned DHCP address 172.25.1.57, mask 255.255.255.0
Router1#
```

20.3.3 Discussion

Cisco started to include DHCP client functionality in IOS Version 12.1(2)T. This allows routers to obtain interface IP address information via DHCP. While we don't recommend using dynamic addressing for routers in an internal network, this can be extremely useful for routers that connect to the Internet through an ISP. It is increasingly common for service providers to use DHCP to give address information to allocate information to client devices.

When an interface on the router is configured as a DHCP client like this, it is able to dynamically learn its IP address, netmask, and default gateway via DHCP. You can also configure the router to accept domain name and DNS server information through DHCP.

In the following screen capture, the router has learned its default route via DHCP. The router displays this DHCP route as a static route and assigns it an administrative distance of 254. This ensures that the default address learned via DHCP is the absolutely last possible route, and any other static or dynamic routes will take precedence:

```
Router1#show ip route
Codes: C - connected, S - static, I - IGRP, R - RIP, M - mobile, B - BGP
       D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
```

```
        N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
        E1 - OSPF external type 1, E2 - OSPF external type 2, E - EGP
        i - IS-IS, L1 - IS-IS level-1, L2 - IS-IS level-2, ia - IS-IS inter area
        * - candidate default, U - per-user static route, o - ODR
        P - periodic downloaded static route

Gateway of last resort is 172.25.1.1 to network 0.0.0.0

     172.25.0.0/24 is subnetted, 1 subnets
C       172.25.1.0 is directly connected, Ethernet0
S*   0.0.0.0/0 [254/0] via 172.25.1.1
Router1#
```

In the ISP situation, the end devices will also need domain name and DNS server information. You can see this information with the *show host* command. This example shows domain name and DNS server information learned via DHCP:

```
Router1#show host
Default domain is oreilly.com
Name/address lookup uses domain service
Name servers are 255.255.255.255, 172.25.1.1

Host                      Port  Flags      Age Type   Address(es)
www.oreilly.com           None  (temp, OK)  0   IP    192.168.22.57
Router1#
```

Note that the router dynamically learned about the domain name and name server information via DHCP, but this information will not overwrite statically configured information. For example, if you manually configure the router with a domain name, the router will quietly ignore the one that it learns through DHCP. The router will simply add any name servers that it learns through DHCP to the static list of manually configured name servers.

The *show ip interface* command tells you that the router learned IP address from DHCP:

```
Router1#show ip interface
Ethernet0 is up, line protocol is up
  Internet address is 172.25.1.57/24
  Broadcast address is 255.255.255.255
  Address determined by DHCP
  MTU is 1500 bytes
  <removed for brevity>
```

Cisco routers do not currently include a way to specify which options the router will request from the server. You also can't see the address lease time remaining. Hopefully Cisco will include these options in a future IOS release.

Although controlling your router addresses from a centralized DHCP server might seem like a good idea, we don't generally recommend it. Routers are the core architecture of a network and should never rely on an external server to obtain IP addressing. Unless a DHCP server is available on every segment,

the router needs a DHCP proxy, which is usually another router with a hardcoded IP address. In disaster scenarios in which many routers fail simultaneously, it can be extremely difficult to bootstrap the network back into operation.

So, except for specific circumstances (such as using a router at the edge of the network to connect to an ISP), we strongly discourage using this DHCP client functionality.

Top

Recipe 20.4 Dynamically Allocating Client IP Addresses via DHCP

20.4.1 Problem

You want to configure your router to be a DHCP server and allocate dynamic IP addresses to client workstations.

20.4.2 Solution

The following set of configuration commands allows the router to dynamically allocate IP addresses to client workstations:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#service dhcp
Router1(config)#ip dhcp pool 172.25.1.0/24
Router1(dhcp-config)#network 172.25.1.0 255.255.255.0
Router1(dhcp-config)#default-router 172.25.1.1
Router1(dhcp-config)#exit
Router1(config)#ip dhcp excluded-address 172.25.1.1 172.25.1.50
Router1(config)#ip dhcp excluded-address 172.25.1.200 172.25.1.255
Router1(config)#end
Router1#
```

20.4.3 Discussion

Cisco incorporated DHCP server functionality starting in IOS Version 12.0(1)T. This allows routers to dynamically allocate IP addresses to client workstations without needing a centralized DHCP server.

Providing DHCP services from a router has some interesting advantages over using a central server. First, distributing the DHCP functionality into the access routers of a large network reduces the risk of a server configuration problem affecting the entire corporate network. Second, maintaining DHCP services within each remote branch router reduces the utilization over expensive WAN links. Third, when a WAN link fails, workstations are still able to function on the local segment. And fourth, the router-based DHCP services can easily be administered by your network engineers via one common interface, the IOS prompt.

In our example, we have configured the router to dynamically allocate IP addresses for the subnet `172.25.1.0/24`. First, we define a range of IP addresses using the *network* command. Then we need to

exclude some addresses from the dynamic range with the *ip dhcp exclude-address* command. If you do not exclude a range of addresses, the router will assign IP addresses from the entire subnet (254 addresses in total). But in most networks, at least some of the addresses are reserved for other purposes.

At a minimum, you should exclude the router's own IP address to prevent the DHCP service from trying to assign it elsewhere, causing a conflict. Many network engineers allocate the lower portion of the subnet to static devices such as routers, servers, printers, and other devices that do not support DHCP. Here, we reserved the first 50 addresses for statically addressed devices (`172.25.1.1` to `172.25.1.50`). We also chose to reserve addresses from `172.25.1.200` to `172.25.1.255` for future use (and to show that you can reserve multiple blocks of addresses).

When DHCP is enabled, the router will allocate IP addresses dynamically by binding them to device MAC addresses in the configured pool. IP addresses are allocated for a defined period of time called a *lease*, which we discuss further in Recipe 20.6. You can view the address bindings with the *show ip dhcp binding* command:

```
Router1#show ip dhcp binding
IP address       Hardware address       Lease expiration        Type
172.25.1.51      0100.0103.85e9.87      Apr 10 2003 08:55 PM    Automatic
172.25.1.52      0100.50da.2a5e.a2      Apr 10 2003 09:00 PM    Automatic
172.25.1.53      0100.0103.ea1b.ed      Apr 10 2003 08:58 PM    Automatic
Router1#
```

The MAC addresses in this output look a little funny because they have two extra hex digits (one extra octet) at the start. We explain why in Recipe 20.7.

The router maintains an internal database of assigned IP addresses and their associated MAC addresses. We show how to write this database to a remote server for backup purposes in Recipe 20.8.

This example dynamically configures DCHP clients with the default router setting using the *default-router* command. We discuss these DHCP options in detail in Recipe 20.5.

20.4.4 See Also

Recipe 20.5; Recipe 20.6; Recipe 20.7; Recipe 20.8; Recipe 20.9

Top

Recipe 20.5 Defining DHCP Configuration Options

20.5.1 Problem

You want to dynamically deliver configuration parameters to client workstations.

20.5.2 Solution

You can configure a wide variety of DHCP parameters for configuring client workstations:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#ip dhcp pool ORAserver
Router1(dhcp-config)#host 172.25.1.34 255.255.255.0
Router1(dhcp-config)#client-name bigserver
Router1(dhcp-config)#default-router 172.25.1.1 172.25.1.3
Router1(dhcp-config)#domain-name oreilly.com
Router1(dhcp-config)#dns-server 172.25.1.1 10.1.2.3
Router1(dhcp-config)#netbios-name-server 172.25.1.1
Router1(dhcp-config)#netbios-node-type h-node
Router1(dhcp-config)#option 66 ip 10.1.1.1
Router1(dhcp-config)#option 33 ip 24.10.1.1 172.25.1.3
Router1(dhcp-config)#option 31 hex 01
Router1(dhcp-config)#lease 2
Router1(dhcp-config)#end
Router1#
```

20.5.3 Discussion

The strength of DHCP is its ability to configure client workstations from a centralized location using DHCP options. It greatly reduces support costs if workstations can dynamically learn all of their configuration options instead of having to send a technician to every desk.

DHCP can assign default routes, domain names, name server addresses, and WINS server addresses, to name just a few. RFC 2132 defines a large number of standard configurable options and includes provisions for further vendor-specific options. However, most networks only use a small subset of these options. To make configuration easier, Cisco provides human-readable names for several of the most common options, as shown in Table 20-2.

Table 20-2. The RFC 2132 equivalent option numbers to Cisco's DHCP commands

Custom name	RFC 2132 option number	Description
client-name	option 12	Hostname (static map only)
default-router	option 3	Default router(s)
domain-name	option 15	Domain name
dns-server	option 6	Name server(s)
netbios-name-server	option 44	WINS server(s)
netbios-node-type	option 46	Netbios node type
lease	option 58	Half of the lease time
host	option 1	Subnet mask (plus IP address)

However, since it would be impossible to create user-friendly names for every possible DHCP option, Cisco allows you to manually configure any option by its number, using the *option* command.

You can also use the option command instead of the custom name. For example, option 6 in RFC 2132 is reserved for name server addresses. Instead of using the Cisco-provided, user-friendly command *dns-server*, as we did in the recipe example, we can define it manually:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#ip dhcp pool 172.25.2.0/24
Router1(dhcp-config)#option 6 ip 172.25.1.1
Router1(dhcp-config)#end
Router1#
```

The router will then translate the manual entry to its user-friendly equivalent in its running configuration file. It can be a little confusing when you look at the router configuration and it isn't what you typed, but both forms are completely equivalent.

Some options will accept multiple entries. For example, the default router option and the *dns-server* option will both accept up to eight IP addresses, in order of preference. However, you will rarely require that many possible entries for a single option. For the default router option in particular, we recommend using defining a single default router address. If there are several routers on a segment, the default router would be the HSRP address. You would use multiple default routers only if you have many routers, but are not running HSRP. This is not a design that we would generally endorse. For more information about HSRP, see Chapter 22.

To make configuration easier, you can create a hierarchy of DHCP pools. Parent DHCP pools are determined by IP address ranges. For instance, in the following example we configure a parent DHCP

pool called *ROOT*, which is used to assign options to the entire classful network range, `172.25.0.0/16`. We then configure two other DHCP pools for specific subnets of `172.25.1.0/24` and `172.25.2.0/24`. These two child pools will automatically inherit the options defined within the *ROOT* pool. You can then overwrite some of the inherited options within the child pools, if necessary:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#ip dhcp pool ROOT
Router1(dhcp-config)#network 172.25.0.0 255.255.0.0
Router1(dhcp-config)#domain-name oreilly.com
Router1(dhcp-config)#dns-server 172.25.1.1 10.1.2.3
Router1(dhcp-config)#lease 2
Router1(dhcp-config)#exit
Router1(dhcp)#ip dhcp pool 172.25.1.0/24
Router1(dhcp-config)#network 172.25.1.0 255.255.255.0
Router1(dhcp-config)#default-router 172.25.1.1
Router1(dhcp-config)#exit
Router1(dhcp)#ip dhcp pool 172.25.2.0/24
Router1(dhcp-config)#network 172.25.2.0 255.255.255.0
Router1(dhcp-config)#default-router 172.25.2.1
Router1(dhcp-config)#lease 0 0 10
Router1(dhcp-config)#end
Router1#
```

> The DHCP lease period is the only option that cannot be inherited from parent DHCP pools. This means that you must explicitly define a lease period for each pool. The router will use the default lease period of one day for any pool that doesn't have its own value.

The example in the solution section of this recipe also includes several *option* statements to define parameters that don't have convenient mnemonics:

```
Router1(dhcp-config)#option 66 ip 10.1.1.1
Router1(dhcp-config)#option 33 ip 192.0.2.1 172.25.1.3
Router1(dhcp-config)#option 31 hex 01
```

These option codes are defined in RFC 2132. In this case, option 66 identifies a TFTP server, option 33 specifies static routes, and option 31 tells the client to use ICMP Router Discovery Protocol (IRDP).

The static route statement tells the end device to send all traffic destined to the host, `192.0.2.1/32`, to the router at `172.25.1.3`.

IRDP allows the client workstation to listen for periodic updates from local routers to determine its default gateway. IRDP is discussed in the introduction to Chapter 22.

20.5.4 See Also

Recipe 20.4; Chapter 22; RFC 2132

Top

Recipe 20.6 Defining DHCP Lease Periods

20.6.1 Problem

You want to change the default lease period.

20.6.2 Solution

To modify the default DHCP lease time for a pool of IP addresses, use the *lease* configuration command:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#ip dhcp pool 172.25.2.0/24
Router1(dhcp-config)#lease 2 12 30
Router1(dhcp-config)#end
Router1#
```

20.6.3 Discussion

The *lease* command takes up to three options: *lease days [hours] [minutes]* with hours and minutes being optional. You can specify a maximum period of 365 days, 23 hours and 59 minutes, and a minimum of 1 second. The default is 1 day.

The shorter the lease period, the faster you can reconfigure DHCP options that may need to change. Short lease periods also permit IP addresses to be returned to the address pool for reallocation more quickly. This can be useful in environments where a large number of end devices frequently connect and disconnect, such as in a public wireless network at an airport. A short lease period of, say, 30 minutes might be useful to ensure that IP addresses are returned quickly to the shared pool. However, short lease periods also mean that workstations must renew their leases more often, which puts an extra strain on the network and DHCP server.

Conversely, a small office with a stable workforce may choose to increase their lease periods. Long lease periods can also reduce the impact of DHCP server failures. Unless a workstation reboots or needs to disconnect and reconnect to the network, most clients will wait until the lease is half expired before asking the server to renew it. If the server is unavailable, the client device will periodically retry the lease renewal until it succeeds. But most organizations have redundant DHCP servers, so there are few real benefits to extremely long lease periods.

In most situations, the default lease period of one day is sufficient. It allows the administrators to change global options in a timely fashion without putting an unnecessary burden on the network or server.

You can also configure the router to assign addresses with infinite lease periods by using the *infinite* keyword:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#ip dhcp pool COOKBOOK
Router1(dhcp-config)#lease infinite
Router1(dhcp-config)#end
Router1#
```

Assigning an infinite lease period removes one of the major advantages of using DHCP. It can be extremely useful to be able to use DHCP to make wholesale configuration changes, but this means that the end devices have to check in periodically to renew their leases. Providing an indefinite lease period largely circumvents this advantage, since it forces you to wait until the end device disconnects and reconnects to the network before you can give it new information. Use the *infinite* keyword with caution.

You can view the lease expiration times of your active clients with the *show ip dhcp binding* command:

```
Router1#show ip dhcp binding
IP address       Hardware address        Lease expiration        Type
172.25.1.33      0100.0103.85e9.87       Infinite                Manual
172.25.1.53      0100.0103.ea1b.ed       Apr 11 2003 08:58 PM    Automatic
172.25.1.57      0100.6047.6c41.a4       Apr 11 2003 09:17 PM    Automatic
Router1#
```

Please refer to Recipe 20.7 for an explanation of the odd-looking MAC addresses in this output.

20.6.4 See Also

Recipe 20.4; Recipe 20.7

Top

Recipe 20.7 Allocating Static IP Addresses with DHCP

20.7.1 Problem

You want to ensure that your router assigns the same IP address to a particular device every time it connects.

20.7.2 Solution

The following commands ensure that the router assigns the same IP address to a device each time it requests one:

```
Router1(config)#ip dhcp pool IAN
Router1(dhcp-config)#host 172.25.1.33 255.255.255.0
Router1(dhcp-config)#client-identifier 0100.0103.85e9.87
Router1(dhcp-config)#client-name win2k
Router1(dhcp-config)#default-router 172.25.1.1
Router1(dhcp-config)#domain-name oreilly.com
Router1(dhcp-config)#dns-server 172.25.1.1
Router1(dhcp-config)#end
Router1#
```

20.7.3 Discussion

The router allows you to statically bind an IP address to a MAC address, which ensures that a particular device always receives the same IP address. This is particularly useful for devices such as servers that must be available for access via a well-known IP address or DNS entry. Any device that accepts inbound sessions will probably require a static address, and allocating these addresses via DHCP provides administrators with greater control over their networks.

The configuration for a static DHCP mapping is slightly different from that of a dynamic pool. In particular, you must assign a separate *dhcp pool* for each static server. In our example, we created a pool named *IAN* to allocate a static IP address to user Ian. Also, instead of defining a network range of IP addresses, you can assign a specific IP address using the *host* command. To avoid address conflicts, make sure that the static address you assign is not part of an already configured dynamic pool. You may need to use the *excluded-address* command for this.

You must configure the static pool with the device's MAC address using the *client-identifier* command. The client identifier is made up of two parts: the media type and the MAC address. The media type

numbers can be found in RFC 1700 ("Assigned Numbers") under the heading "Number Hardware Type." For 10/100/1000Mb Ethernet, the media type number is `01`. The router will combine the media type and MAC address into one large address and automatically add the dots if you don't type them.

From this point on, the router will accept the same options as the dynamic pool options. Options can be inherited from dynamic pools as well. To view the configured static binding use the *show ip dhcp binding* command:

```
Router1#show ip dhcp binding
IP address       Hardware address       Lease expiration        Type
172.25.1.33      0100.0103.85e9.87      Infinite                Manual
172.25.1.52      0100.50da.2a5e.a2      Apr 11 2003 09:00 PM    Automatic
172.25.1.53      0100.0103.ea1b.ed      Apr 11 2003 08:58 PM    Automatic
Router1#
```

This shows that we have successfully mapped a static IP address of `172.25.1.33` to MAC address `0001-0385-e987`. DHCP has put the code `01` at the start of this address to indicate that it is an Ethernet MAC.

Note that the router marks the static clients as "Manual" to differentiate them from the others. Although the output indicates that the static lease is indefinite, in reality it is not. The static *dhcp pool* can be assigned any lease period you desire, but the router will only allocate the single static address. This can be useful if you want to change other DHCP options for this end device.

20.7.4 See Also

Recipe 20.5; RFC 1700

Top

Recipe 20.8 Configuring a DHCP Database Client

20.8.1 Problem

You want to back up your DHCP database of address assignments to another device so that you won't lose it if the router reloads.

20.8.2 Solution

You can ensure that your DHCP address assignments are not lost when a router reloads by configuring the router to periodically copy its DHCP database to a remote server.

The first example configures a router to use FTP to copy the DHCP database to a remote server:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#ip dhcp database ftp://dhcp:bindsave@172.25.1.1/dhcp-leases
Router1(config)#end
Router1#
```

The second example uses TFTP as the transport protocol:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#ip dhcp database tftp://172.25.1.1/dhcp-leases
Router1(config)#end
Router1#
```

The third configures RCP as the transport protocol:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#ip dhcp database rcp://dhcp@172.25.1.1/dhcp-leases
Router1(config)#end
Router1#
```

20.8.3 Discussion

By default, the router stores its DHCP binding database in memory. So, when the router reloads, all DHCP database information is lost. You can configure the router to periodically send a copy of this database to a remote server to avoid losing this important information. If the router recycles for any

reason, it will automatically load the last version of the database file from the remote server and proceed from where it left off.

In our example, we have configured our test router to store its DHCP database to the server at `172.25.1.1` via FTP. FTP uses a simple username and password authentication system, so we have included the user ID *dhcp* and password *bindsave* in the command. After authenticating, the router will store its database in a file called *dhcp-leases* in the default directory for this user ID. If you have multiple routers storing their databases on the same server, make sure to use a unique file name that includes the router name. This can assist in troubleshooting later.

To view the status of the DHCP database, use the *show ip dhcp database* command:

```
Router1#show ip dhcp database
URL        : ftp://dhcp:bindsave@172.25.1.1/dhcp-leases
Read      : Never
Written   : Apr 09 2003 10:24 PM
Status    : Last write succeeded. Agent information is up-to-date.
Delay     : 300 seconds
Timeout   : 300 seconds
Failures  : 1
Successes:  30

Router1#
```

Note that the router is storing the database to the URL we specified, and that its last write was successful. The output also tells us that the router has successfully written its database to the server 30 times and only experienced a single failure. All of this indicates that the DHCP backup is working well. If we log on to the server and view the contents of the database file, we will see the current DCHP bindings:

```
Freebsd% cat dhcp-leases
*time* Apr 09 2003 10:24 PM

!IP address     Type  Hardware address   Lease expiration
172.25.1.52     id    0100.50da.2a5e.a2  Apr 10 2003 09:00 PM
172.25.1.53     id    0100.0103.ea1b.ed  Apr 10 2003 08:58 PM

!IP address     Interface-index  Lease expiration       Vrf
*end*
Freebsd%
```

Fortunately, the file is in human-readable form and lets us see the latest DHCP bindings. Static address bindings never change, so they are not sent to the remote database server.

We will simulate a power failure by reloading the router:

```
Router1#show ip dhcp database
URL        : ftp://dhcp:bindsave@172.25.1.1/dhcp-leases
Read      : Apr 10 2003 10:35 PM
Written   : Never
```

```
Status   : Last read succeeded. Bindings have been loaded in RAM.
Delay    : 300 seconds
Timeout  : 300 seconds
Failures : 0
Successes: 1

Router1#
```

Now when we display the database information, we see that the router successfully read the database from the server and loaded the bindings into memory. Viewing the router bindings now, you can see that the database has been recovered:

```
Router1#show ip dhcp binding
IP address       Hardware address       Lease expiration        Type
172.25.1.33      0100.0103.85e9.87      Infinite                Manual
172.25.1.52      0100.50da.2a5e.a2      Apr 10 2003 09:00 PM    Automatic
172.25.1.53      0100.0103.ealb.ed      Apr 10 2003 08:58 PM    Automatic
Router1#
```

20.8.4 See Also

Recipe 20.4; Recipe 20.10

Top

Recipe 20.9 Configuring Multiple DHCP Servers per Subnet

20.9.1 Problem

You want to configure multiple routers to act as DHCP servers for the same subnet to ensure availability.

20.9.2 Solution

You can configure multiple routers to act as DHCP servers for a single subnet by ensuring that they don't use the same pool of addresses.

Router1:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#ip dhcp pool 172.22.1.0/24
Router1(dhcp-config)#network 172.22.1.0 255.255.255.0
Router1(dhcp-config)#default-router 172.22.1.1
Router1(dhcp-config)#domain-name oreilly.com
Router1(dhcp-config)#dns-server 172.25.1.1 10.1.2.3
Router1(dhcp-config)#exit
Router1(config)#ip dhcp excluded-address 172.22.1.1 172.22.1.49
Router1(config)#ip dhcp excluded-address 172.22.1.150 172.22.1.254
Router1(config)#ip dhcp database ftp://dhcp:bindsave@172.25.1.1/dhcp-leases-rtr1
Router1(config)#end
Router1#
```

Router2:

```
Router2#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router2(config)#ip dhcp pool 172.22.1.0/24
Router2(dhcp-config)#network 172.22.1.0 255.255.255.0
Router2(dhcp-config)#default-router 172.22.1.1
Router2(dhcp-config)#domain-name oreilly.com
Router2(dhcp-config)#dns-server 172.25.1.1 10.1.2.3
Router2(dhcp-config)#exit
Router2(config)#ip dhcp excluded-address 172.22.1.1 172.22.1.149
Router2(config)#ip dhcp database ftp://dhcp:bindsave@172.25.1.1/dhcp-leases-rtr2
Router2(config)#end
Router2#
```

20.9.3 Discussion

You can configure multiple DHCP servers to service the same subnet; in fact, we recommend it. However, you must ensure that the two servers do not share the same dynamic pool of IP addresses.

In our example, we chose to let Router1 allocate the addresses between `172.25.1.50` and `172.25.1.149`, while Router2 allocates the pool from `172.25.1.150` to `172.25.1.254`. We reserved the range from `172.25.1.1` to `172.25.1.50` for static IP addresses.

You need to map out the allocated address space ahead of time and use the *ip dhcp excluded-address* configuration command to ensure that there is no overlap. The critical thing about this type of configuration is to ensure that the allocated pool of DHCP addresses on a single router is sufficiently large to handle all of the DHCP clients on the segment if the other router fails. This way, although the two routers do not allocate addresses from the same pool (which would cause problems for the DHCP databases), you still have complete redundancy.

We have also configured the routers to store their DHCP binding databases to separate files on the same server. For added protection, you could opt to store the individual database files to different servers.

The rest of the option settings are identical, although they don't necessarily have to be. You might want to change the name server order, for instance, to help balance the load between DNS servers.

20.9.4 See Also

Recipe 20.4; Recipe 20.5

Top

Recipe 20.10 Showing DHCP Status

20.10.1 Problem

You want to display the status of the DHCP server functions on the router.

20.10.2 Solution

To display the IP address bindings and their associated leases, use this command:

```
Router1#show ip dhcp binding
```

The following command displays any IP address conflicts that the router has detected in the DHCP address pool:

```
Router1#show ip dhcp conflict
```

You can view the status of remote database backups with this command:

```
Router1#show ip dhcp database
```

And you can see the global DHCP server statistics like this:

```
Router1#show ip dhcp server statistics
```

20.10.3 Discussion

To display the status of the DHCP service, use the *show ip dhcp* EXEC command. If you add the keyword *binding*, this command displays the current DHCP bindings, which include the assigned IP addresses, the associated client MAC addresses, and the lease expiration time:

```
Router1#show ip dhcp binding
IP address       Hardware address       Lease expiration        Type
172.25.1.51      0100.0103.85e9.87      Apr 10 2003 08:55 PM    Automatic
172.25.1.52      0100.50da.2a5e.a2      Apr 10 2003 09:00 PM    Automatic
172.25.1.53      0100.0103.ea1b.ed      Apr 10 2003 08:58 PM    Automatic
Router1#
```

To view the IP addresses that are currently in conflict, use the *conflict* keyword. This command displays all of the IP addresses that the router has discovered conflicts for, and how the conflict was discovered:

```
Router1#show ip dhcp conflict
IP address        Detection method   Detection time
172.25.1.51       Ping               Apr 09 2003 09:08 PM
172.25.1.54       Gratuitous ARP     Apr 09 2003 10:00 PM
Router1#
```

With the *database* keyword, you can view the database configuration and status. This command shows all of the configured remote database servers and the current read and write status reports:

```
Router1#show ip dhcp database
URL         : ftp://dhcp:bindsave@172.25.1.1/dhcp-leases
Read      : Never
Written   : Apr 09 2003 10:24 PM
Status    : Last write succeeded. Agent information is up-to-date.
Delay     : 300 seconds
Timeout   : 300 seconds
Failures  : 1
Successes:  30

Router1#
```

Finally, the *statistics* keyword lets you view the overall DHCP statistics:

```
Router1#show ip dhcp server statistics
Memory usage          17996
Address pools         4
Database agents       1
Automatic bindings    2
Manual bindings       1
Expired bindings      3
Malformed messages    0

Message               Received
BOOTREQUEST           0
DHCPDISCOVER          63
DHCPREQUEST           203
DHCPDECLINE           1
DHCPRELEASE           27
DHCPINFORM            19

Message               Sent
BOOTREPLY             0
DHCPOFFER             63
DHCPACK               139
DHCPNAK               2
Router1#
```

This command displays high-level DHCP statistics, including the number of bindings, and address pools, as well as the number of sent and received messages.

20.10.4 See Also

Recipe 20.4

Top

Recipe 20.11 Debugging DHCP

20.11.1 Problem

You want to debug a DHCP problem.

20.11.2 Solution

To debug the server events, use the following EXEC command:

```
Router1#debug ip dhcp server events
```

The following command will allow you to monitor the actual DHCP-related packets being transmitted and received by the router:

```
Router1#debug ip dhcp server packet
```

20.11.3 Discussion

The following debug capture shows a router performing normal housekeeping duties such as updating its address pools, checking for expired leases, assigning new leases, and revoking expired leases:

```
Router1#debug ip dhcp server events
Sep 15 00:58:17.218: DHCPD: returned 172.25.1.51 to address pool COOKBOOK
Sep 15 00:58:22.566: DHCPD: assigned IP address 172.25.1.51 to client 0100.0103.85e9.
87.
Sep 15 01:01:15.056: DHCPD: writing bindings to ftp://dhcp:bindsave@172.25.1.1/dhcp-
leases-rtr1.
Sep 15 01:01:15.132: DHCPD: writing address 172.25.1.51.
Sep 15 01:01:15.148: DHCPD: wrote automatic bindings to ftp://dhcp:bindsave@172.25.1.
1/dhcp-leases-rtr1.
Sep 15 01:01:58.816: DHCPD: checking for expired leases.
Sep 15 01:03:58.841: DHCPD: checking for expired leases.
Sep 15 01:05:58.859: DHCPD: checking for expired leases.
Sep 15 01:07:58.874: DHCPD: checking for expired leases.
Sep 15 01:09:58.885: DHCPD: checking for expired leases.
Sep 15 01:09:58.885: DHCPD: the lease for address 172.25.1.51 has expired.
Sep 15 01:09:58.885: DHCPD: returned 172.25.1.51 to address pool COOKBOOK.
```

The next debug capture shows a typical DHCP client transaction between the client and router. The net result is that the router assigns IP address `172.25.1.51` to MAC address `0001.0385.e987`:

```
Router1#debug ip dhcp server packet
Sep 15 01:19:41.211: DHCPD: DHCPDISCOVER received from client 0100.0103.85e9.87 on
interface  FastEthernet0/0.1.
```

```
Sep 15 01:19:43.212: DHCPD: Sending DHCPOFFER to client 0100.0103.85e9.87 (172.25.1.
51).
Sep 15 01:19:43.212: DHCPD: creating ARP entry (172.25.1.51, 0001.0385.e987).
Sep 15 01:19:43.212: DHCPD: unicasting BOOTREPLY to client 0001.0385.e987 (172.25.1.
51).
Sep 15 01:19:43.216: DHCPD: DHCPREQUEST received from client 0100.0103.85e9.87.
Sep 15 01:19:43.216: DHCPD: Sending DHCPACK to client 0100.0103.85e9.87 (172.25.1.
51).
Sep 15 01:19:43.216: DHCPD: creating ARP entry (172.25.1.51, 0001.0385.e987).
Sep 15 01:19:43.216: DHCPD: unicasting BOOTREPLY to client 0001.0385.e987 (172.25.1.
51).
Router1#
```

Top

Chapter 21. NAT

Top

Introduction

Network Address Translation (NAT) was first described in RFC 1631 in 1994. The authors of that document were trying to solve the imminent problem of running out of IPv4 addresses. They proposed a simple but brilliant solution: allow devices on the inside of a network to use the standard pool of unregistered IP addresses currently defined in RFC 1918. The router or firewall at the boundary between the internal private network and the external public network could then use software to rewrite the internal IP addresses of every packet, replacing them with valid registered addresses.

There are four kinds of addresses: *inside local*, *inside global*, *outside local*, and *outside global*. Inside and outside are relative terms if you're just connecting two private networks. But if you are connecting a private network to the public Internet, the Internet is considered the outside. A local address is generally the private address, while the global address is the globally unique public address.

To help make these terms more clear, suppose you are connecting a network that uses RFC 1918 private addresses to the public Internet. Inside your network you have private addresses, such as `192.168.1.0/24`. These are the inside local addresses. NAT will translate these addresses to globally unique registered addresses, which are also the inside global addresses. The addresses on the public Internet are outside global addresses. These external network addresses are all registered in this case, so there is no need to translate them. If translation was needed, an outside global address would be changed to an outside local address.

To put this another way, the address that internal devices use to communicate with other internal devices is the inside local address. The address that internal devices uses to communicate with external devices is the outside local address. The address that external devices uses to communicate with internal devices is the inside global address. Finally, external devices communicate with one another using outside global addresses.

NAT makes it possible to have a huge internal network with thousands of local addresses represented by a handful (or perhaps even just one) global address. This is why NAT is often credited with alleviating the address shortage problem. But it solves this problem only if most people who use it have more local than global addresses.

In practice, NAT offers a huge range of possibilities. You can map local addresses uniquely to individual global addresses. You can share one global address among several local addresses. You can allocate global addresses from a pool as they are requested, or have a single global address and map all local addresses to this one address. You can even define a combination of these different alternatives.

When a device sends a packet out from the private to the public network, the translator replaces the local source address with a registered address, then routes the packet. For an inbound packet, the

translator replaces the global address with the local address and routes the packet into the internal network. The translator has a much more difficult job with inbound packets than outbound, because it has to figure out which internal device to send the packet to. Since many internal devices may be using the same global address, the translator has to keep a state table of all of the devices that send or receive packets to or from the external network.

Suppose, for example, that two internal users are both using HTTP to view information on the public network. The translator must be able to determine which packets are intended for which internal device. It's not sufficient to simply look at the external device's IP address—both of these users could be looking at the same web page. They would both wind up with severely scrambled screens if the translator couldn't tell which packets to send to which internal user.

This particular example is made somewhat easier by the fact that HTTP uses TCP. Because TCP is a connection-based protocol, well-defined TCP session initiation and termination helps the translator to sort out the inbound flows. In this case, Cisco's NAT implementation uses Port Address Translation (PAT), which means that the router rewrites the source port numbers, and uses the new values as tags to distinguish between the two flows. Because UDP also uses port numbers, the same PAT technique also works here, although the router doesn't keep the translation table entries active for the same length of time.

ICMP, on the other hand, is considerably more difficult for NAT to keep straight. For example, if two internal users both ping the same external site at the same time, the translator has to assume that it will receive the responses in the same order that they were sent. Fortunately, this rarely causes real problems in production networks. But it is worth remembering that, whether it can use PAT or not, NAT requires the router to keep track of a lot of state information that routers don't usually care about.

Further, because IP addresses and port numbers are included in both IP and TCP checksums, the translator must recalculate these checksum values for every packet. So NAT always consumes more CPU and memory on the router or firewall that it runs on. This resource usage increases rapidly with both the number of packets and the number of different flows.

The other important thing to remember about NAT is that some protocols include IP address information in the payload of the packet, as well as in the IP header. For example, the ubiquitous FTP protocol has a PORT command that contains an IP address encoded in ASCII. In this case, the FTP protocol is well understood and NAT implementations can look out for the PORT command. But in other, less popular protocols, strange problems can occur. And, if a server happens to run FTP on a nonstandard TCP port, you must tell NAT about the change so that it can rewrite the payload addresses.

SNMP also includes IP addresses in packet payloads. For example, IP address information is part of the standard interface MIB because the address is an important piece of information about the interface. However, rewriting addresses in the payloads of SNMP packets is a much more difficult problem than finding the IP address for FTP, because the address could be anywhere in the payload. It is also possible for the addresses in the payload to refer to different interfaces than the address in the header.

And, to make the problem even more difficult, there is no common standard format for IP addresses in SNMP packets. They are sometimes transmitted as dotted decimal ASCII strings, as packed hex bytes, or in a variety of other formats depending on the specific MIB. Consequently, Cisco routers do not attempt to rewrite IP addresses in the payloads of SNMP packets.

We have also seen custom-built applications that make life very hard for NAT by encoding IP addresses and port numbers in the data segment of a packet, then using this information to attempt new connections. It can be very hard to get NAT to work in cases like this. Often the only workaround is to encapsulate the ill-behaved application in a tunnel.

Top

Recipe 21.1 Configuring Basic NAT Functionality

21.1.1 Problem

You want to set up Network Address Translation on your router.

21.1.2 Solution

In the simplest NAT configuration, all of your internal devices use the same external global address as the router's external interface:

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#access-list 15 permit 192.168.0.0 0.0.255.255
Router(config)#ip nat inside source list 15 interface Ethernet0/0 overload
Router(config)#interface FastEthernet0/0
Router(config-if)#ip address 192.168.1.1 255.255.255.0
Router(config-if)#ip nat inside
Router(config-if)#exit
Router(config)#interface FastEthernet0/1
Router(config-if)#ip address 192.168.2.1 255.255.255.0
Router(config-if)#ip nat inside
Router(config-if)#exit
Router(config)#interface Ethernet0/0
Router(config-if)#ip address 172.16.1.5 255.255.255.252
Router(config-if)#ip nat outside
Router(config-if)#end
Router#
```

21.1.3 Discussion

In this example, the router will rewrite the addresses of all of the internal devices whose IP addresses are in the range `192.168.0.0/16`. When these internal devices connect to devices on the outside of the network, they will all appear to have the same source address as the external interface of the router, `172.16.1.5`.

This example actually includes two internal interfaces and one external interface. You designate the internal interfaces with the *ip nat inside* command. You can have as many inside interfaces as you like:

```
Router(config)#interface FastEthernet0/1
Router(config-if)#ip nat inside
```

You also need to designate at least one outside interface using the *ip nat outside* command:

```
Router(config-if)#interface Ethernet0/0
Router(config-if)#ip nat outside
```

You can also use several outside interfaces, but this configuration can be very difficult to control, so we don't recommend it. Next, configure the actual translation action with the line:

```
Router(config)#ip nat inside source list 15 interface Ethernet0/0 overload
```

This tells the router to translate the source addresses of any internal devices that match access list number 15. The router will translate the source addresses of all of these devices to the address that is configured on the interface `Ethernet0/0`, which is the outside interface.

The *overload* keyword is actually assumed here—if you leave it off, the router will put it in automatically. This option tells the router that many internal devices can use the same global address simultaneously. We explain this option in more detail in Recipe 21.2.

To help explain what the access list in this command does, we change it to include every address in the range except one:

```
Router(config)#access-list 15 deny 192.168.1.101
Router(config)#access-list 15 permit 192.168.0.0 0.0.255.255
```

If you make a connection from the excluded address (`192.168.1.101`) after issuing this command, the router will not rewrite its internal address. Instead, this address will appear unchanged on the outside.

NAT can be quite confusing because people usually think that there are firewall functions associated with it. There are none. If you exclude one device from your NAT access list as we just discussed, anybody on the outside of the network will be able to connect to this internal device by using its real address. Further, there is nothing to prevent an inbound packet from reaching a particular internal device if the person on the outside knows the real internal address and can route to it.

Top

Recipe 21.2 Allocating External Addresses Dynamically

21.2.1 Problem

You want to dynamically select addresses from a pool.

21.2.2 Solution

You can configure the router to automatically select global addresses from a pool as they are required:

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#access-list 15 permit 192.168.0.0 0.0.255.255
Router(config)#ip nat pool NATPOOL 172.16.1.100 172.16.1.150  netmask 255.255.255.0
Router(config)#ip nat inside source list 15 pool NATPOOL
Router(config)#interface FastEthernet 0/0
Router(config-if)#ip address 192.168.1.1 255.255.255.0
Router(config-if)#ip nat inside
Router(config-if)#exit
Router(config)#interface FastEthernet 0/1
Router(config-if)#ip address 192.168.2.1 255.255.255.0
Router(config-if)#ip nat inside
Router(config-if)#exit
Router(config)#interface Ethernet1/0
Router(config-if)#ip address 172.16.1.2 255.255.255.0
Router(config-if)#ip nat outside
Router(config-if)#end
Router#
```

21.2.3 Discussion

This example is similar to Recipe 21.1. The important functional difference is that the internal devices will appear on the outside with different global addresses. The first internal device that makes an outbound connection will get the first address in the range (`172.16.1.100`), the next one will get the next address (`172.16.1.101`), and so forth.

Configure the range with the *ip nat pool* command:

```
Router(config)#ip nat pool NATPOOL 172.16.1.100 172.16.1.150  netmask 255.255.255.0
Router(config)#ip nat inside source list 15 pool NATPOOL
```

In this case, the *ip nat inside* command does not have the *overload* keyword. Without this keyword,

when the pool of addresses is used up, the router will respond to any additional requests with an "ICMP host unreachable" message. Any additional devices that try to make connections through this router will simply fail. But if you include the *overload* keyword, the router will simply start over at the beginning of the range and allocate multiple interior addresses for each external one:

```
Router(config)#ip nat inside source list 15 pool NATPOOL overload
```

Once again, as in Recipe 21.1, any devices that are excluded by the access list will simply not use this NAT rule. The excluded devices will appear on the outside with their real (inside local) IP addresses.

In this example, the IP address of the external interface is `172.16.1.2/24`, and the pool of translation external addresses for use in translation is `172.16.1.100` through `172.16.1.150`. So the pool of NAT addresses is part of the same IP subnet as the external IP address of the NAT router. This is a common practice for Internet connections in which the ISP assigns a range of global addresses, but it is not necessary.

Your NAT pool can be anything, as long as the external network knows that this router can route to the NAT addresses. This is particularly useful in cases where you need a larger pool than what is available in that one subnet. We could easily have made our NAT pool span a Class A range such as `10.0.0.0/8`, giving us access to a huge number of external addresses. Of course, this range is not globally unique, so it can't be used on the public Internet:

```
Router(config)#ip nat pool NATPOOL 10.0.0.1 10.255.255.254  netmask 255.0.0.0
```

21.2.4 See Also

Recipe 21.1

Top

Recipe 21.3 Allocating External Addresses Statically

21.3.1 Problem

You want to translate specific internal IP addresses to specific external addresses.

21.3.2 Solution

For some applications, you need each internal (inside local) address to always translate to the same external (inside global) address. This is particularly true if you need inbound connections from the outside network to always reach a particular internal device, such as a web or email server:

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#ip nat inside source static 192.168.1.15 172.16.1.10
Router(config)#ip nat inside source static 192.168.1.16 172.16.1.11
Router(config)#interface FastEthernet 0/0
Router(config-if)#ip address 192.168.1.1 255.255.255.0
Router(config-if)#ip nat inside
Router(config-if)#exit
Router(config)#interface FastEthernet 0/1
Router(config-if)#ip address 192.168.2.1 255.255.255.0
Router(config-if)#ip nat inside
Router(config-if)#exit
Router(config)#interface Ethernet1/0
Router(config-if)#ip address 172.16.1.2 255.255.255.0
Router(config-if)#ip nat outside
Router(config-if)#end
Router#
```

21.3.3 Discussion

This recipe includes static translations for two internal devices. The internal address `192.168.1.15` will always appear on the outside as `172.16.1.10`, and `192.168.1.16` will always appear as `172.16.1.11`. Note that because these translations are static, they will work in either direction. Any packets sent to the NAT address from the external network will reach the internal device, and external devices can even initiate TCP sessions.

This example performs NAT translation only for these two specific addresses. The router will route all other addresses normally without any address translation.

21.3.4 See Also

Recipe 21.4

Top

Recipe 21.4 Translating Some Addresses Statically and Others Dynamically

21.4.1 Problem

You want certain hosts to have static address translation properties and all others to use dynamic translation.

21.4.2 Solution

In some cases, you might need to use a combination of the two approaches. Some internal devices will always translate to specific external addresses, but others will use a dynamic pool. This is often the case when you have a few internal servers that need to be accessed from outside of the network, but the other devices will make only outbound connections:

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#access-list 15 deny 192.168.1.15 0.0.0.0
Router(config)#access-list 15 deny 192.168.1.16 0.0.0.0
Router(config)#access-list 15 permit 192.168.0.0 0.0.255.255
Router(config)#ip nat inside source static 192.168.1.15 172.16.1.10
Router(config)#ip nat inside source static 192.168.1.16 172.16.1.11
Router(config)#ip nat pool NATPOOL 172.16.1.100 172.16.1.150  netmask 255.255.255.0
Router(config)#ip nat inside source list 15 pool NATPOOL overload
Router(config)#interface FastEthernet0/0
Router(config-if)#ip address 192.168.1.1 255.255.255.0
Router(config-if)#ip nat inside
Router(config-if)#exit
Router(config)#interface FastEthernet0/1
Router(config-if)#ip address 192.168.2.1 255.255.255.0
Router(config-if)#ip nat inside
Router(config-if)#exit
Router(config)#interface Ethernet0/0
Router(config-if)#ip address 172.16.1.2 255.255.255.0
Router(config-if)#ip nat outside
Router(config-if)#end
Router#
```

21.4.3 Discussion

In this recipe, we have used the same pool of dynamic addresses as in Recipe 21.2, combined with the same two static translations from Recipe 21.3. It is often useful to combine NAT techniques like this,

particularly when you use the connection between these networks for several different applications. Some applications might need to work with well-known IP addresses, while others could work well from a dynamic pool.

The access list in this example specifically excludes the two addresses that will use static (rather than dynamic) NAT. This is not strictly necessary because the static NAT commands appear to have precedence over dynamic NAT in the router. However, this is still a good practice because your intentions are absolutely clear to anybody looking at the router configuration.

The other important thing to note in this example is that we have explicitly removed the static NAT addresses from the dynamic NAT pool. The dynamic pool is from `172.16.1.100` to `172.16.1.150`, while the static addresses are `172.16.1.10` and `172.16.1.11`. This is critically important because the dynamic NAT allocation does not check each address in the pool to make sure that is not configured for static NAT translation. You could have serious address conflicts if you do not explicitly separate the static from the dynamic NAT addresses.

21.4.4 See Also

Recipe 21.2; Recipe 21.3

Top

Recipe 21.5 Translating in Both Directions Simultaneously

21.5.1 Problem

You want to translate both internal and external addresses.

21.5.2 Solution

In some cases, you might need to translate IP addresses on both sides of your router:

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#access-list 15 deny 192.168.1.15
Router(config)#access-list 15 permit 192.168.0.0 0.0.255.255
Router(config)#access-list 16 deny 172.16.5.25
Router(config)#access-list 16 permit 172.16.0.0 0.0.255.255
Router(config)#ip nat pool NATPOOL 172.16.1.100 172.16.1.150  netmask 255.255.255.0
Router(config)#ip nat pool INBOUNDNAT 192.168.15.100 192.168.15.200  netmask 255.255.25
Router(config)#ip nat inside source list 15 pool NATPOOL overload
Router(config)#ip nat inside source list 16 pool INBOUNDNAT overload
Router(config)#ip nat inside source static 192.168.1.15 172.16.1.10
Router(config)#ip nat outside source static 172.16.5.25 192.168.15.5
Router(config)#ip route 192.168.15.0 255.255.255.0  Ethernet0/0
Router(config)#interface FastEthernet 0/0
Router(config-if)#ip address 192.168.1.1 255.255.255.0
Router(config-if)#ip nat inside
Router(config-if)#exit
Router(config)#interface FastEthernet 0/1
Router(config-if)#ip address 192.168.2.1 255.255.255.0
Router(config-if)#ip nat inside
Router(config-if)#interface Ethernet0/0
Router(config-if)#ip address 172.16.1.2 255.255.255.0
Router(config-if)#ip nat outside
Router(config-if)#end
Router#
```

21.5.3 Discussion

Sometimes you need to translate IP addresses on both the inside and the outside interfaces. This might happen, for example, when you need to connect to another network that uses an overlapping range of unregistered addresses. Cisco routers can do NAT translations of addresses on both the external and internal interfaces at the same time.

In this case, the router will rewrite external addresses in the range `172.16.0.0/16` so that they appear

to be on the `192.168.15.0/24` subnet in the range specified by the *INBOUNDNAT* pool. And, at the same time, it will rewrite internal addresses that are part of the `192.168.0.0/16` subnet so that they will appear on the outside to be part of `172.16.1.0/24` in the range specified by the *NATPOOL* pool.

Note that the access lists defining which addresses should use the dynamic address pool both refer to the real addresses (inside local and outside global). So, for internal devices, the access list should refer to the real internal addresses, while the list for external devices should refer to the real external addresses.

The most significant reason for using this feature is to remove a conflict due to overlapping address ranges. The following example shows how to remove an address conflict at the router between two networks that both use the ubiquitous `10.0.0.0/8` address range. We will map the outside network to `11.0.0.0/8` and the inside network to `12.0.0.0/8` . Note that these two address ranges are both registered network numbers, so doing this will cause some problems for Internet access. We recommend doing this only as a temporary measure to resolve an IP address conflict caused by merging two networks with overlapping IP address ranges:

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#access-list 17 permit 10.0.0.0 0.255.255.255
Router(config)#access-list 18 permit 10.0.0.0 0.255.255.255
Router(config)#ip nat pool OUTPOOL 11.0.0.1 11.255.255.254  netmask 255.0.0.0 type
match-host
Router(config)#ip nat pool INPOOL 12.0.0.1 12.255.255.254  netmask 255.0.0.0 type
match-host
Router(config)#ip nat inside source list 17 pool INPOOL
Router(config)#ip nat outside source list 18 pool OUTPOOL
Router(config)#ip route 11.0.0.0 255.0.0.0  Ethernet0/0
Router(config)#ip route 12.0.0.0 255.0.0.0  FastEthernet1/0
Router(config)#interface FastEthernet1/0
Router(config-if)#ip address 10.1.1.1 255.255.255.0
Router(config-if)#ip nat inside
Router(config-if)#exit
Router(config)#interface Ethernet0/0
Router(config-if)#ip address 10.2.1.2 255.255.255.0
Router(config-if)#ip nat outside
Router(config-if)#end
Router#
```

Note that we have used the match-host keyword in the NAT pool definitions:

```
Router(config)#ip nat pool OUTPOOL 11.0.0.1 11.255.255.254  netmask 255.0.0.0 type
match-host
```

When you use this option, the router will translate the network prefixes and leave the host portions of the address intact. So, in this example, the arbitrary IP address `10.1.2.3` would become `11.1.2.3` . Only the first byte would be changed. The key advantage of this method is that the translations are always the same, so you can reliably make connections between any internal and external devices in

either direction. You cannot do this with the ordinary dynamic address pools that we have discussed so far. Note that the *overload* option makes no sense in this configuration.

There are a few important things to watch out for when using NAT in both directions. First, the router must have routing table entries for the fictitious IP addresses. It is quite likely that the translated addresses used for external devices will not be part of a physical IP network that the router knows how to reach. This is why we have configured a static route directing traffic for this range out through the external interface:

```
Router(config)#ip route 11.0.0.0 255.255.255.0  Ethernet0/0
```

The second important thing to remember is that with dynamic NAT, the router does not create a translation for each device until it needs to. If you want to connect through the router to a particular translated address, you must make sure that the router retains the translation table information. This means that if you want any-to-any connections in either direction, you must use either static mappings or the *match-host* keyword. Dynamic NAT will not allow access in both directions.

The third important thing to remember is that all of the other routers must know how to reach the translated addresses. So, if the external network is translated from `10.0.0.0/8` to `11.0.0.0/8` , then you need to make sure that the internal routers all know that they can reach this fictitious `11.0.0.0/8` network through the NAT router. The best way to do this is by simply redistributing the static routes for the fictitious networks through your dynamic routing protocol.

Recipe 21.6 shows a somewhat better way to solve this overlapping address problem. Instead of doing simultaneous translation in both directions on the same router, it is better to do it on two routers with a different, nonconflicting address range in the middle. One router will simply translate the prefix for one of these networks from `10.0.0.0/8` to `11.0.0.0/8` . The other router will translate the addresses on the other network from `10.0.0.0/8` to `12.0.0.0/8` . This is a much more stable solution, and it does not suffer from the problems of dynamic NAT mentioned earlier.

21.5.4 See Also

Recipe 21.1 ; Recipe 21.2 ; Recipe 21.3 ; Recipe 21.4 ; Recipe 21.6

Top

Recipe 21.6 Rewriting the Network Prefix

21.6.1 Problem

You want to rewrite all of the addresses in a particular range by simply replacing the prefix with one of equal length.

21.6.2 Solution

Sometimes you need to connect your network to another network that uses an unregistered range, such as `172.16.0.0/16`. However, if you already use this range in your network, the easiest thing to do is to simply replace this prefix with another one that doesn't have a conflict, such as `172.17.0.0/16`:

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#ip nat outside source static network 172.16.0.0 172.17.0.0 /16  no-
alias
Router(config)#ip route 172.16.0.0 255.255.0.0  Ethernet1/0
Router(config)#ip route 172.17.0.0 255.255.0.0  Ethernet1/0
Router(config)#interface FastEthernet 0/0
Router(config-if)#ip address 10.1.1.1 255.255.255.0
Router(config-if)#ip nat inside
Router(config-if)#exit
Router(config)#interface Ethernet1/0
Router(config-if)#ip address 172.16.1.6 255.255.255.252
Router(config-if)#ip nat outside
Router(config-if)#end
Router#
```

21.6.3 Discussion

Unlike the previous examples, this recipe shows a very simple static form of NAT that translates addresses by simply replacing one prefix with another. So, for example, the remote host, `172.16.55.19`, gets its address rewritten as `172.17.55.19`.

The router can accomplish this with the following command:

```
Router(config)#ip nat outside source static network 172.16.0.0 172.17.0.0 /16  no-
alias
```

This defines a static mapping of one network prefix to another, as required.

Note that we have included the *no-alias* keyword in this command. If we didn't include this keyword, the router would try to generate aliases for the translated addresses to allow it to answer ARP requests for them. This keyword is necessary because one of the router's own interfaces belongs to the translated range.

Top

Recipe 21.7 Adjusting NAT Timers

21.7.1 Problem

You want to change the length of time that NAT entries remain active.

21.7.2 Solution

The router will keep NAT entries in the translation table for a configurable length of time. For TCP connections, the default timeout period is 86,400 seconds, or 24 hours. Because UDP is not connection-based, the default timeout period is much shorter: only 300 seconds (5 minutes). The router will remove translation table entries for DNS queries after only 60 seconds.

You can adjust these parameters using the *ip nat translation* command, which accepts arguments in seconds:

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#ip nat translation tcp-timeout 500
Router(config)#ip nat translation udp-timeout 30
Router(config)#ip nat translation dns-timeout 30
Router(config)#ip nat translation icmp-timeout 30
Router(config)#ip nat translation finrst-timeout 30
Router(config)#ip nat translation syn-timeout 30
Router(config)#end
Router#
```

To save router memory, you can also define a maximum number of NAT translation table entries:

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#ip nat translation max-entries 1000
Router(config)#end
Router#
```

21.7.3 Discussion

There are many reasons for adjusting these various timeout parameters; most are related to router performance. If sessions are generally short-lived, it is a waste of memory to maintain the NAT entries for a long time. The *finrst-timeout* and *syn-timeout* parameters are also useful when the router is connected to the public Internet because they can help to prevent DoS attacks that are based on sending

TCP control packet such as SYN, ACK, and FIN. If the router keeps only the NAT entries associated with these packets for a brief period of time, you can help to limit the impact of such attacks.

We recommend using extreme caution with the *max-entries* command:

```
Router(config)#ip nat translation max-entries 1000
```

When you set a limit like this, the router will reject any additional attempts to use NAT. So, in this example, if you already had 1000 NAT table entries, the router would simply drop any new connection attempts. This can be a useful way to prevent excessive NAT processing from overloading the router, but it can also block legitimate access.

It is difficult to select a useful upper limit to the size of the NAT table in general. In most cases it is best to use the default, which does not enforce any upper limit. You should use this command only if you start to run into serious memory or CPU utilization problems. Because restricting the table size tells the router to refuse any further requests, this method should be a last resort. In most cases it is more effective to decrease the various timeout values as shown in this recipe.

Start by looking at your NAT translation table (as shown in Recipe 21.9), and see what most of the entries look like. If you are using the *overload* option, you may find that there are several different entries for each internal host, each for different port numbers or protocols. The relatively long 24-hour timeout period for TCP sessions is probably the best place to start. Decreasing the size of the NAT table by reducing this timeout period will not cause any application problems.

21.7.4 See Also

Recipe 21.9

Top

Recipe 21.8 Changing TCP Ports for FTP

21.8.1 Problem

You have an FTP server using a non-standard TCP port number.

21.8.2 Solution

The FTP protocol includes IP address information in the packet payload. Normally, Cisco's NAT implementation rewrites IP address information in the payloads of FTP packets by looking in every packet sent on TCP port 21, which is the port that FTP uses to pass session control information by default. So when an FTP server uses a nonstandard TCP port number for session control, you must configure the NAT router to expect FTP packets on the new port number:

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#access-list 19 permit 192.168.55.5
Router(config)#ip nat service list 19 ftp tcp port 8021
Router(config)#ip nat service list 19 ftp tcp port 21
Router(config)#end
Router#
```

21.8.3 Discussion

As we mentioned in the introduction to this chapter, the common FTP protocol includes IP address information in the packet payload. Cisco routers expect this, and rewrite the information appropriately. But some FTP servers use a nonstandard TCP port number, which means that NAT will break the protocol. So, in IOS Version 11.3, Cisco introduced the ability to look for FTP payload information on alternate TCP port numbers.

The example configures the router to expect FTP packets for the server `192.168.55.5` on both the default port number `21` and the nonstandard port number `8021`. You can easily configure similar commands for other servers as well, or expand the access list to include several servers that all use the same nonstandard FTP port number.

Top

Recipe 21.9 Checking NAT Status

21.9.1 Problem

You want to see the current NAT information.

21.9.2 Solution

There are several useful EXEC commands for checking the status of NAT on a router. You can view the NAT translation table using the following command:

```
Router#show ip nat translation
```

You can clear all or part of the NAT translation table by specifying either an asterisk (*) or a particular address. To clear a specific entry, you must specify either the global address for a device that is inside, or a local address for a device that is outside:

```
Router#clear ip nat translation *
Router#clear ip nat translation inside 172.18.3.2
Router#clear ip nat translation outside 192.168.1.10
```

You will often want to look at NAT statistics, including information on which interfaces use NAT, how many entries are in the NAT table, how often they have been used, and, most importantly, how often packets have bypassed NAT. The command to see this is *show ip nat statistics*:

```
Router#show ip nat statistics
```

You can clear these statistics as follows:

```
Router#clear ip nat statistics
```

21.9.3 Discussion

The NAT translation table contains information about every translation that the router is currently tracking. In this example, there have been two connections between the interior (`192.168.1.10`) and exterior (`172.18.3.2`) device. The first of these connections is shown as ICMP:

```
Router#show ip nat translation
Pro Inside global       Inside local        Outside local      Outside global
icmp 172.16.1.100:21776 192.168.1.10:21776   172.18.3.2:21776    172.18.3.2:21776
tcp 172.16.1.100:1029   192.168.1.10:1029    172.18.3.2:23       172.18.3.2:23
```

```
--- 172.16.1.10          192.168.1.15          ---                   ---
--- 172.16.1.11          192.168.1.16          ---                   ---
Router#
```

This command shows only the currently active NAT table entries. You can see, for example, that it translates the inside local address `192.168.1.10` to the inside global address `172.16.1.100`. But this router isn't configured to translate outside addresses, so the outside local addresses are the same as the outside global addresses. As discussed in Recipe 21.7, the router removes dynamic NAT entries after a defined period of time. By default, the router will delete NAT entries for TCP connections after 24 hours.

The output has five columns. The first is the protocol. This column is blank unless you use the *overload* option in your NAT configuration. The "Inside global" address column is the translated address of an internal device. The "Inside local" column, on the other hand, shows the real internal address for the same device. The "Outside local" column shows the translated addresses of external devices, while "Outside global" shows their real addresses.

This can be a little bit confusing at first sight. The real address on the inside is "local," and the translated address is "global," while the real address on the outside is "global," and it is translated to a "local" address. You can resolve this confusion by remembering that global addresses are always on the outside, and local addresses are always on the inside.

The last two rows represent simple static NAT entries. It shows, for example, that the internal device whose real address is `192.168.1.15` is translated to `172.16.1.10` when its packets pass through this router. There are no external addresses listed for this entry. Because it is a static entry, this translation is the same for any external device. However, the row immediately above this one shows all four entries:

```
tcp 172.16.1.100:1029  192.168.1.10:1029  172.18.3.2:23       172.18.3.2:23
```

This line includes a lot of useful information. The first column indicates that this row represents a TCP connection, and that the translation is a dynamic entry. On the inside, the source address is `192.168.1.10` and the source TCP port is `1029`, while the destination is `172.18.3.2` and the destination port is `23`. On the outside, the destination address and port are the same, but the source address is rewritten as `172.16.1.100` and the source port is `1029`.

The *verbose* keyword makes this command show age information about each table entry:

```
Router#show ip nat translation verbose
Pro Inside global       Inside local         Outside local      Outside global
icmp 172.16.1.100:21776 192.168.1.10:21776    172.18.3.2:21776    172.18.3.2:21776
192.168.3.2:4235
    create 00:00:36, use 00:00:36, left 00:00:23, flags: extended
tcp 172.16.1.100:1029  192.168.1.10:1029  172.18.3.2:23       172.18.3.2:23
    create 00:00:15, use 00:00:13, left 00:00:46, flags: extended, timing-out
--- 172.16.1.10        192.168.1.15       ---                   ---
    create 1d00h, use 00:23:08, flags: static
--- 172.16.1.11        192.168.1.16       ---                   ---
    create 1d00h, use 00:15:28, flags: static
```

```
Router#
```

This level of detail is most useful when you are trying to diagnose NAT table timeout issues.

The *show ip nat statistics* command includes useful information about the translation configuration. The following example shows one external and two internal interfaces, with a dynamic NAT pool that runs from `172.16.1.100` to `172.16.1.150`:

```
Router#show ip nat statistics
Total active translations: 3 (2 static, 1 dynamic; 1 extended)
Outside interfaces:
  Ethernet0/0
Inside interfaces:
  FastEthernet0/0, FastEthernet0/1
Hits: 2628  Misses: 44
Expired translations: 37
Dynamic mappings:
-- Inside Source
access-list 15 pool NATPOOL refcount 1
 pool NATPOOL: netmask 255.255.255.0
        start 172.16.1.100 end 172.16.1.150
        type generic, total addresses 2, allocated 1 (50%), misses 9
Router#
```

The "Hits" field shows the total number of times that the router has had to create new translation table entries. The "Misses" field counts the exceptions. In this case, there is an access list that excludes certain internal IP addresses.

Top

Recipe 21.10 Debugging NAT

21.10.1 Problem

You want to debug a NAT problem.

21.10.2 Solution

Cisco routers include a simple but useful debug facility for NAT. The basic form of the command is *debug ip nat*:

```
Router#debug ip nat
```

You can also add the *detailed* keyword to this command to get more information on each NAT event:

```
Router#debug ip nat detailed
```

It is often useful to use an access list with the debug command. You can do this by simply specifying the number of the access list. This will allow you to look only at NAT events for particular IP addresses that are permitted by the access list:

```
Router#debug ip nat 15
```

You can also combine an access list with the *detailed* keyword for more focused debugging:

```
Router#debug ip nat 15 detailed
```

21.10.3 Discussion

The following shows some typical log entries:

```
Router#terminal monitor
Router#debug ip nat
Sep   8 19:51:08.396 EDT: NAT: s=192.168.3.1->192.168.19.1, d=192.168.3.2 [0]
Sep   8 19:51:11.560 EDT: NAT*: s=192.168.1.10->192.168.19.55, d=192.168.3.2 [4909]
Sep   8 19:51:11.568 EDT: NAT*: s=192.168.3.2, d=192.168.19.55->192.168.1.10 [4909]
Sep   8 19:51:11.572 EDT: NAT: s=192.168.3.2, d=192.168.19.55->192.168.1.10 [4909]
Sep   8 19:51:12.552 EDT: NAT*: s=192.168.1.10->192.168.19.55, d=192.168.3.2 [4911]
Sep   8 19:51:12.564 EDT: NAT*: s=192.168.3.2, d=192.168.19.55->192.168.1.10 [4911]
```

This particular trace follows a simple series of *ping* packets. The interior device `192.168.1.10` sends

ICMP *ping* packets to the external destination `192.168.3.2`. The router rewrites the internal address as `192.168.19.55` and forwards the packet to the external destination.

You can also see the *ping* responses coming back from the destination device. The router rewrites the internal address back to its true value and forwards the packet appropriately.

Top

Chapter 22. Hot Standby Router Protocol

Top

Introduction

Hot Standby Router Protocol (HSRP) is a Cisco proprietary standard that allows a router on a LAN segment to automatically take over if another one fails. It was developed to solve a common problem in shared networks such as Ethernet or Token Ring. The devices on these shared network segments are usually configured with a single default gateway address that points to the router that connects to the rest of the network. The problem is that even if there is a second router on the segment that is also capable of being the default gateway, the end devices don't know about it. Therefore, if the first default gateway router fails, the network stops working.

Many methods for addressing this problem have come and gone over the years. The most obvious and most seriously flawed solution is to have the end users reconfigure the default gateway address in their workstations. This is a terrible solution for several reasons. There is a large chance of typographical errors: the conversion is slow, laborious, and often requires a reboot of the workstation; it relies on users noticing the problem in a timely manner, and it is unlikely that anyone will bother changing the address back when the original router recovers; it also requires that a human is handy to make the change, but devices such as printers and servers don't usually have anyone sitting beside them when problems appear.

A slightly better solution that many organizations have used is to run a dynamic routing protocol such as RIP or OSPF directly on the servers and workstations. Unix-based operating systems have access to good routing protocol implementations such as the *routed* and *gated* programs. However, many popular desktop and server operating systems do not support these protocols. Even if every device in the network could run a routing protocol, this is not a very good solution to the problem for several reasons. Routing protocols tend not to converge well when the number of devices gets too large. So this technique would, at the very least, require a major network redesign. It is also a generally bad idea to let end devices affect the global routing tables throughout the network. If one of these devices is not configured properly, it could cause serious routing problems. And, more philosophically, it is a good principle of network design to keep network functions on network devices. Workstations and servers already have enough to do without having to perform a router's job as well.

ICMP Router Discovery Protocol (IRDP), which is described in RFC 1256, represents still another interesting idea for allowing end devices to find a new router when their default gateway fails. This protocol requires routers to periodically send multicast "hello" messages to the LAN segment. End devices listen for these messages and use them to build their internal routing tables. If an end device doesn't hear these hello messages for a while, it assumes that the router must have failed. The end device then sends a multicast query looking for a new router to take over. Again, this method requires special software on the end devices. Few devices support IRDP, and it has never enjoyed particularly wide acceptance.

Cisco routers do support IRDP; enable it using the *ip irdp* interface command:

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#interface FastEthernet 0/1
Router(config-if)#ip irdp
Router(config-if)#end
Router#
```

We do not recommend using IRDP, however, because it is unlikely that all of the devices on a segment will be able to use and react to it appropriately. HSRP, which we will turn to in a moment, provides a much more robust and flexible router redundancy mechanism.

As an aside, there is another protocol also called Inter-Domain Routing Protocol (IDRP), which is part of the OSI protocol suite that provides similar functionality to BGP. The similarity of the names is just an accident. There is no relation between these protocols, although it is easy to get the acronyms confused.

One of the more popular solutions to the problem of router redundancy uses *Proxy ARP*, which is enabled by default on Cisco routers. In this configuration, the end devices are not configured with a default gateway at all. Instead, they discover the path to remote devices the same way that they find devices on the local LAN segment: by using Address Resolution Protocol (ARP). With Proxy ARP, the routers will respond to ARP requests on behalf of remote device. Then the originating device simply sends a packet to the remote destination IP address using the MAC address of the local router, which is exactly the desired behavior.

The problem with the Proxy ARP solution is that it doesn't switch to a backup router very quickly when the primary router fails. End devices don't change their MAC addresses very often, and the whole ARP cache procedure assumes that if an entry was once valid, it will remain valid unless it is explicitly changed by means of a *gratuitous ARP* from the other device declaring a new address. Most devices will remove a stale ARP entry if the device fails to respond for several minutes, but this is clearly not fast enough for a reliable failover mechanism. The only ways to speed this procedure up are to reboot or manually clear the ARP cache on the end device. Proxy ARP is also a rather messy solution because it requires a potentially large number of ARP requests on the local segment. Because ARP requests are broadcasts, this can cause serious problems on a busy segment.

Cisco developed Hot Standby Router Protocol (HSRP) to address the problem of router redundancy in a more reliable way. It provides a non-disruptive automatic failover method that doesn't require end devices to run any special software. HSRP is documented in RFC 2281, although it is a Cisco proprietary standard that is not implemented by other vendors.

HSRP works by allowing two routers to share the same virtual IP and MAC addresses. End devices simply send their off-segment packets to these addresses as a standard default gateway. One of the routers will receive and forward the packets, so either can fail without disrupting traffic flow. One

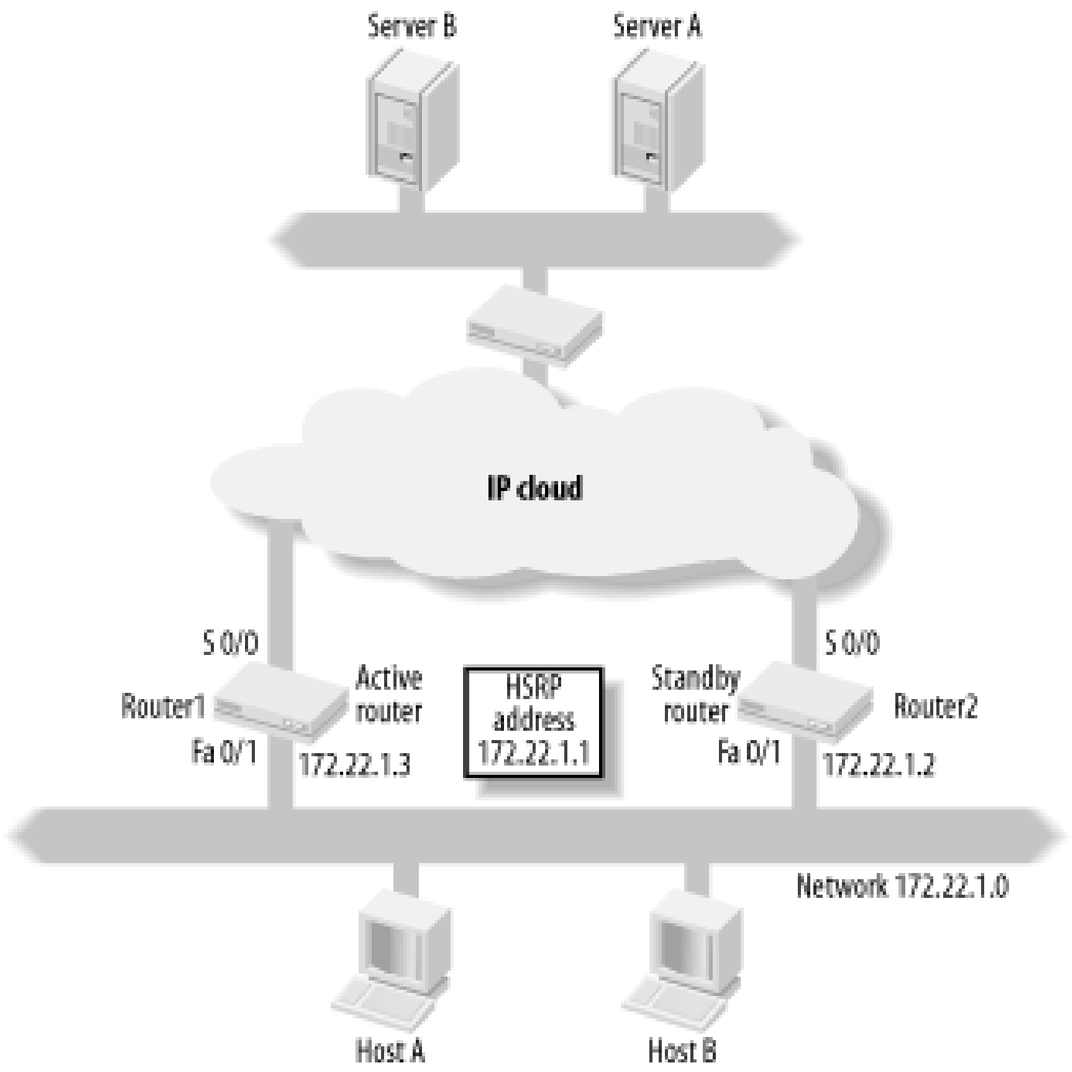
router is always active, and the other acts as a standby in case the first one should fail. In fact, you can configure many standby routers for extreme high-availability situations. The HSRP routers that share a virtual IP address send multicast packets back and forth periodically. If the primary router ever stops sending these packets for any reason, one of the standby routers immediately takes over both the IP and MAC addresses, and continues to forward packets.

Another similar solution to the same problem is the open standard Virtual Router Redundancy Protocol (VRRP), which is defined in RFC 2338. VRRP is currently supported by many vendors, but has not yet become an official IETF standard. However, because Cisco had already developed HSRP when VRRP was announced, most Cisco devices do not implement VRRP. We say most because Cisco does have some VLAN switches that support VRRP. This leads us to suspect that Cisco may eventually provide VRRP on other products. However, because they continue to add useful features to HSRP, it seems unlikely that VRRP will ever completely replace HSRP on Cisco equipment:

Figure 22-1 shows a simple example of an HSRP network that will make a good reference point for many of the examples in this chapter. If Host A uses Router1 as its default gateway, then it will lose access to the network if Router1 fails. This is true even if there is a second router, Router2, on the same segment.

In the HSRP configuration shown in this diagram, Router1 and Router2 share the virtual IP address `172.22.1.1`. These two routers also have their own IP addresses, `172.22.1.3` and `172.22.1.2`, respectively. This is a relatively common and useful way of allocating IP addresses in a `/24` network. All end devices use the `.1` address for their default gateway, which is the virtual router address. The two physical routers then use the `.2` and `.3` addresses for their real addresses.

Figure 22-1. A HSRP-enabled network segment



HSRP sends multicast packets between routers on the common LAN segment using multicast address `224.0.0.2` and UDP port `1985`. By default, these packets are exchanged every 3 seconds, and if they are not seen for 10 seconds, the standby router takes over. Each router in a group has a priority that defines whether it is active or standby. Both the timers and the priority values are configurable.

You can use up to 256 HSRP groups, numbered 0 through 255, on Ethernet and FDDI type networks. This can be useful in network designs in which a central backbone connects many distinct network segments carrying different subnets. For Token Ring LANs, however, you can only configure three distinct HSRP groups, numbered 0 through 2. For Token Rings, you can configure additional groups if you use the Burned In Address (BIA) on the router's Token Ring port (as discussed in Recipe 22.7). The limitation of three HSRP groups applies to the default configuration mode, which uses a common MAC address for the virtual IP address on both routers.

It is important to note that the HSRP group number is significant only on the local LAN segment. You can use the same group number on different interfaces on the same router if the segments do not connect. However, many network administrators find that it helps to avoid confusion if they use different group numbers on different interfaces. Recipe 22.4 shows a good example of a case in which having multiple HSRP groups on a single LAN segment is extremely useful.

For Ethernet LANs, HSRP uses a standard set of MAC addresses from the range allocated to Cisco. The virtual Ethernet MAC addresses are `00-00-0C-07-AC-XX`, where the `XX` represents the HSRP group number in hex (00-FF). The following output shows an HSRP packet captured using the popular Ethereal packet analyzer package:

```
Ethernet II
```

```
     Destination: 01:00:5e:00:00:02
     Source: 00:00:0c:07:ac:01
    Type: IP (0x0800)
Internet Protocol
    Version: 4
    Header length: 20 bytes
        Total Length: 48
        Protocol: UDP (0x11)
    Source: 172.22.1.3
    Destination: ALL-ROUTERS.MCAST.NET (224.0.0.2)
User Datagram Protocol
    Source port: 1985 (1985)
     Destination port: 1985 (1985)
    Length: 28
Cisco Hot Standby Router Protocol
    Version: 0
    Op Code: Hello (0)
    State: Active (16)
     Hellotime: Default (3)
     Holdtime: Default (10)
    Priority: 120
    Group: 1
    Reserved: 0
    Authentication Data: Non-Default (OREILLY)
     Virtual IP Address: 172.22.1.1
```

Token Ring LANs use so-called functional MAC addresses, which are reserved for special-purpose applications. HSRP uses `C0-00-00-01-00-00`, `C0-00-00-02-00-00`, and `C0-00-00-04-00-00` for groups 0, 1, and 2, respectively. However, as discussed in Recipe 22.7, many organizations actually use the BIA of the Token Ring interface card with HSRP instead of these functional addresses. As we mentioned earlier, when you use the BIA, you can configure additional groups. But it's important to remember that they will all use the same MAC address. This is useful only when you want to configure several IP subnets on the same physical ring and use HSRP on all of them.

HSRP is only used for IP networking. However, the fact that it allows two devices to use the same MAC address can cause serious problems for some other protocols. In particular, if you use DECnet or XNS on the same segment, you must use the BIA to avoid bad protocol interactions. The command for this is *use-bia*, which we discuss in Recipe 22.7.

Although HSRP represents a useful alternative to Proxy ARP, as we have already mentioned, you can use them together. This is particularly useful when you are migrating from an old Proxy ARP configuration to HSRP. In this case, the router uses the HSRP virtual MAC address when it responds to ARP requests.

It is also worth noting that the router will disable ICMP redirects by default when you enable HSRP. Normally, when you have two routers on the same segments, ICMP redirection allows you to send a packet to either one. If the other router has a better path to the destination, the receiving router will forward the packet to the other router and send a special ICMP redirect packet back to the source

device. The source device receives this packet and updates its internal routing table accordingly so that all future packets to this destination will use the better router.

Normally, you don't want to use ICMP redirection with HSRP because it would allow the end devices to learn the real physical MAC addresses for the routers. Since the end devices update their internal routing tables with this information, a failure by one of the routers would prevent the other from taking over all routing functions.

However, in Recipe 22.5, we show how to configure HSRP routers to use ICMP redirection so that they use only the HSRP virtual MAC address instead of any physical addresses.

Top

Recipe 22.1 Configuring Basic HSRP Functionality

22.1.1 Problem

You want a backup router to take over the MAC and IP addresses of a primary router if the primary fails.

22.1.2 Solution

Figure 22-1 represents a typical network design for use with HSRP on an Ethernet type LAN segment (including FastEthernet, Gigabit Ethernet and 10-Gigabit Ethernet). There are two routers called Router1 and Router2, which have the IP addresses `172.22.1.3` and `172.22.1.2`, respectively. When both routers are available, we want Router1 to handle all of the traffic using the virtual IP address `172.22.1.1`.

Configure the first router as follows:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#interface FastEthernet 0/1
Router1(config-if)#ip address 172.22.1.3 255.255.255.0
Router1(config-if)#standby 1 ip 172.22.1.1
Router1(config-if)#standby 1 priority 120
Router1(config-if)#end
Router1#
```

The second router's configuration is similar, except that the interface has a different real IP address and a lower HSRP priority level:

```
Router2#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router2(config)#interface FastEthernet 1/0
Router2(config-if)#ip address 172.22.1.2 255.255.255.0
Router2(config-if)#standby 1 ip 172.22.1.1
Router2(config-if)#standby 1 priority 110
Router2(config-if)#end
Router2#
```

22.1.3 Discussion

In this example, we use the first address of the subnet, `172.22.1.1`, as the virtual HSRP address, and

consequently the default gateway for the segment. This is a relatively common practice and a good rule of thumb because it makes troubleshooting easier. Whatever segment you are looking at, you always know that the first address in the range is the default gateway.

For HSRP configurations, we recommend using the next two addresses as the physical addresses (`172.22.1.2` and `172.22.1.3` in the example). This way, when you are looking at a problem, you always tell exactly what the physical router addresses should be so you can *ping* them or log in and check their configurations.

In fact, you can use physical addresses that are from a different IP subnet than the virtual address. However, we don't recommend this because, once again, it can make troubleshooting problems extremely difficult, particularly if the HSRP configuration is broken so that neither router can adopt the right virtual address.

You can also use HSRP with secondary IP addresses, but we don't recommend this unless it is unavoidable. With modern VLAN-based network designs, secondary IP addresses on a LAN segment should be used only temporarily, such as when you are making addressing changes on your network. To configure a secondary HSRP address, use the *secondary* keyword.

```
Router2(config-if)#standby 1 ip 172.22.2.1 secondary
```

The number "1" following all of the *standby* commands in this recipe is a group number. You can leave this out, in which case the router will assume group 0. The group number is necessary if you have more than one pair of HSRP routers on the same segment. However, if a router runs HSRP on more than one interface, many administrators also find that it helps with troubleshooting if they configure different group numbers for each interface. This is particularly true if the different segments appear as different VLANs in the same switch. Since the default virtual MAC address depends only on group number, it can cause problems for some switches to see the same MAC address on two different VLANs.

You must configure all of the routers that share the same virtual IP address with the same group number. For Ethernet type interfaces, group numbers can have any value between 0 and 255, while for Token Ring interfaces you can use group numbers 0, 1, or 2 unless you include the *use-bia* command, which we discuss in Recipe 22.7.

The standby priority command, which appears in both routers, is optional. Priority values can be between 0 and 255. If you don't configure a priority, the router will use a default value of 100. We changed the priority on both routers for clarity. However, we highly recommend giving at least one of the routers a non-default priority. If both routers have the same priority, they must elect an active router. RFC 2281, which documents the HSRP protocol, stipulates that the interface with the higher physical IP address will win this election if two routers become active simultaneously. However, in practice, one router almost always comes up first and wins.

You will usually want to give one of the routers a higher priority so that it is active by default. This way you force a particular router to be active, which can help with troubleshooting.

When a router becomes active, it broadcasts a gratuitous ARP packet with the HSRP virtual MAC address to the affected LAN segment. If the segment uses an Ethernet switch, this allows the switch to change the location of the virtual MAC address so that packets go to the new router instead of the one that is no longer active. End devices don't actually need this gratuitous ARP if the routers use the default HSRP MAC address. However, if the routers use the Burned-In Address (BIA), as in Recipe 22.7, the gratuitous ARP is critical for updating the ARP caches of end devices to point to the new router.

By default, the router will send gratuitous ARP packets every 10 seconds. You can adjust this interval with the *standby mac-refresh* command, which takes an argument between 0 and 255 seconds. Many switches will remove an entry from their MAC tables if they don't see at least one packet every 5 minutes (300 seconds). However, sometimes random errors mean that a packet is not received properly. We recommend using a value that is less than 150 seconds, such as 30 seconds:

```
Router2(config-if)#standby mac-refresh 30
```

If you don't want the router to send these packets at all, you can specify a value of 0. Note that this command does not specify a group. If you change this value, it changes for all groups on the interface.

You can use the *show standby* command to see the status of HSRP on a router:

```
Router1#show standby
FastEthernet0/1 - Group 1
  Local state is Active, priority 120
  Hellotime 3 sec, holdtime 10 sec
  Next hello sent in 0.424
  Virtual IP address is 172.22.1.1 configured
  Active router is local
  Standby router is 172.22.1.2 expires in 7.456
  Virtual mac address is 0000.0c07.ac01
  5 state changes, last state change 12:40:42
Router1#
```

In this example, you can see that this router is the active router, so the other must either be in a standby or an unavailable state. If this router were in the standby state, the line that currently says "Active router is local" would show the physical IP address of the active router. In this case, the following line shows the IP address of the standby router. The fact that the standby router is listed with an expiration time means that it is available. If the other router became unavailable for any reason, this line would be replaced by one saying "Standby router is unknown expired." This makes it very easy to see the state of both routers at once.

This command also shows other useful information such as how frequently the router sends HSRP hello packets (every three seconds). The "holdtime" shows how long the routers will wait before switching states, 10 seconds in this case. The output also shows the virtual MAC address that HSRP is using, and that this router has an HSRP priority of 120.

You can also configure a particular shared MAC address if you don't want to use the default or the BIA:

```
Router1(config-if)#standby 1 mac-address 0000.0c07.ad01
```

When you use this option, you should configure the same MAC address on both routers. By default, when you configure a particular standby group, the router will always select the same MAC address. This is useful because it means that two routers will always agree on the MAC address that they will be sharing. But it can also cause confusion for a LAN switch that sees the same MAC address on two different VLANs. This could happen, for example, if you have two routers using HSRP group number 1 on one VLAN and two different routers using the same group number on a different VLAN. If you encounter problems, you can use the *standby mac-address* command to ensure that the HSRP MAC addresses are globally unique.

The recipe example shows two routers for simplicity. You could add more routers to the same HSRP group by simply specifying another unique real IP address and a lower HSRP priority:

```
Router3#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router3(config)#interface FastEthernet 1/0
Router3(config-if)#ip address 172.22.1.4 255.255.255.0
Router3(config-if)#standby 1 ip 172.22.1.1
Router3(config-if)#standby 1 priority 100
Router3(config-if)#end
Router3#
```

However, there is questionable benefit to using more than two routers in a full redundancy configuration. The reason has to do with simple probability. The probability of one router failing is relatively small, but it can happen. Cisco quotes typical Mean Time Between Failure (MTBF) values for its routers between 15 and 20 years. Assuming that it takes a full day to repair a broken router, this means that you should expect to need a backup about 0.018% of the time. This is a very small number, but if you have a lot of routers, the probability of having a critical failure somewhere in your network can become rather large.

If you have a backup router that uses HSRP to automatically and transparently take over all routing functions for this segment, then you will only have a critical outage if both routers fail simultaneously. The probability of this happening is the square of the probability of one router failing, or roughly 3×10^{-6}%. The effective aggregate MTBF has gone from 15 years to about 80,000 years.

The advantage to using a backup router should be obvious. If you then added another backup router to this network segment, the probability of failure becomes about 6×10^{-10}%, for an effective MTBF of over 400,000,000 years. Very few networks actually need that sort of reliability.

In fact, all of these statistical arguments assume that the failure of one router is completely uncorrelated with the failure of the backup. This is not the case if both devices run from the same circuit breaker, for example. So, if you find that you frequently suffer from multiple simultaneous failures, you should probably figure out why the failures are correlated. Simply adding another router might not help the

situation at all.

And, of course, there are other reasons why routers become unavailable. In many networks, the most compelling reason for using HSRP is that it makes routine maintenance possible without disrupting production traffic. This is particularly important in networks that must be available at all times. You might even decide to use three HSRP routers on a segment to ensure that you still have full redundancy even when you take one of the routers down for maintenance.

22.1.4 See Also

Recipe 22.7; RFC 2281

Top

Recipe 22.2 Using HSRP Preempt

22.2.1 Problem

You want to ensure that a particular router is always selected as the "active" HSRP router whenever it is up and functioning.

22.2.2 Solution

You can ensure that a particular router is always selected as the HSRP active router if it is available. On the router that you wish to make your primary active HSRP router, you need to set a higher priority level and use the *standby preempt* command:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#interface FastEthernet 0/1
Router1(config-if)#standby 1 ip 172.22.1.1
Router1(config-if)#standby 1 priority 120
Router1(config-if)#standby 1 preempt
Router1(config-if)#end
Router1#
```

The only difference on the second router is the HSRP priority level. The lower priority in conjunction with the *standby preempt* command forces the second HSRP router to remain in a standby state unless the primary router becomes unreachable. Then, when the primary router becomes available again, the secondary router will relinquish its role and allow the primary router to take over as the active HSRP router:

```
Router2#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router2(config)#interface FastEthernet 1/0
Router2(config-if)#standby 1 ip 172.22.1.1
Router2(config-if)#standby 1 priority 110
Router2(config-if)#standby 1 preempt
Router2(config-if)#end
Router2#
```

You can also configure this feature by specifying the *preempt* keyword on the *standby priority* configuration command. Both methods produce exactly the same results:

```
Router2#configure terminal
```

```
Enter configuration commands, one per line.  End with CNTL/Z.
Router2(config)#interface FastEthernet 1/0
Router2(config-if)#standby 1 ip 172.22.1.1
Router2(config-if)#standby 1 priority 110 preempt
Router2(config-if)#end
Router2#
```

22.2.3 Discussion

The *standby preempt* feature means that the router with the higher priority is always the active router if it is available. If two routers have the same priority, the interface with the highest physical IP address becomes active. However, as we mentioned in Recipe 22.1, it is a good practice to always set the priorities so that one router is the clear winner. This avoids any confusion.

In Recipe 22.3 we describe another extremely useful HSRP feature, called interface tracking, which allows HSRP to react to problems on other interfaces on the router. In this case, it becomes much more important to ensure that the right router carries traffic in the default situation. We strongly advocate using the *standby preempt* feature whenever you use HSRP.

The protocol handles this feature by allowing routers to send HSRP coup packets. If another router receives a coup message or even an HSRP hello message from a router with higher priority, it will immediately send an HSRP resign message in return, and give up its active state. Because one router will always have either higher priority or a higher IP address than any of the others, there is always one router that can unambiguously claim precedence, ensuring that all HSRP coups are bloodless.

In IOS 12.0(9) and higher, you can alter this behavior slightly by making the active router wait before relinquishing its active state. This is sometimes helpful when the newly active router needs a little bit of time to bring up slow WAN interfaces and populate routing tables. In particular, Frame Relay interfaces can take as long as a minute to synchronize LMI (see Chapter 10 for more discussion of Frame Relay). However, by default, HSRP preempt will make the router become the active default gateway for the LAN within seconds of bringing up the LAN interface.

The following command will make a router wait 60 seconds before becoming the active HSRP router:

```
Router2(config-if)#standby 1 preempt delay 60
```

You can set this delay to any value between 0 and 3600 seconds. You can also combine this command with the *priority* command:

```
Router2(config-if)#standby 1 priority 110 preempt delay 60
```

The *show standby* command shows additional information when the preempt option is configured:

```
Router1#show standby
FastEthernet0/1 - Group 1
  Local state is Active, priority 120, may preempt
  Preemption delayed for at least 60 secs
```

```
  Hellotime 3 sec, holdtime 10 sec
  Next hello sent in 2.236
  Virtual IP address is 172.22.1.1 configured
  Active router is local
  Standby router is 172.22.1.2 expires in 9.304
  Virtual mac address is 0000.0c07.ac01
  5 state changes, last state change 12:41:53
Router1#
```

In this case, the router is configured to wait 60 seconds before preempting. If you don't define a delay, the router omits this line from the output.

22.2.4 See Also

Recipe 22.1; Recipe 22.3; Chapter 10

Top

Recipe 22.3 Making HSRP React to Problems on Other Interfaces

22.3.1 Problem

You want HSRP to switch to the backup router when another port on the primary router becomes unavailable.

22.3.2 Solution

The *standby track* configuration command reduces the priority of an active HSRP router into a standby mode when one of its interfaces becomes unavailable. If the priority drops far enough, another router will take over:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#interface FastEthernet0/1
Router1(config-if)#standby 1 ip 172.22.1.1
Router1(config-if)#standby 1 priority 120
Router1(config-if)#standby 1 preempt
Router1(config-if)#standby 1 track Serial0/0 20
Router1(config-if)#end
Router1#
```

22.3.3 Discussion

This configuration option is particularly useful when you have two identically configured WAN access routers using HSRP on their LAN ports. In this case, if you are using a dynamic routing protocol, then losing the WAN connection to one of the routers isn't actually a disaster. The routing protocol will tell the active router to forward all of its outbound traffic to the standby router, which will still have a good connection. However, this is obviously inefficient. It would be better if the active router simply resigned its active status and let the standby router take over.

HSRP does this by decreasing the priority for the active router. It decreases the priority by 10 points by default, but you can configure this amount. In the example, the router drops its HSRP priority by 20 points when the interface `Serial0/0` becomes unavailable:

```
Router1(config-if)#standby 1 track Serial0/0 20
```

In all of our examples so far, we have configured the priorities of the two HSRP routers to have a difference of 10 priority points. If we used the default priority drop in this *standby track* command, a failure of the tracked interface would give the two routers equal priority. The router with the higher IP address will then become the active router when this interface fails, but this might not be the right choice.

We have specified a value of 20 priority points in this command to ensure that the other router will take over appropriately.

You can use the *standby track* command to track any router interface, or even multiple interfaces. To track several interfaces, you just specify all of the interfaces in separate *standby track* commands:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#interface FastEthernet0/1
Router1(config-if)#standby 1 ip 172.22.1.1
Router1(config-if)#standby 1 priority 120
Router1(config-if)#standby 1 preempt
Router1(config-if)#standby 1 track Serial0/0 20
Router1(config-if)#standby 1 track Serial0/1 20
Router1(config-if)#end
Router1#
```

In this example, we have explicitly configured HSRP to decrement the priority by 20 if either of the tracked interfaces fails. So, if both interfaces fail, the priority will drop by 40 points.

For the standby track command to work properly, you must also configure standby preempt , as described in Recipe 22.2. This is because you want to allow the router that now has the higher priority (after the tracked interface failure) to send an HSRP coup message and take control.

When you use tracking like this, the *show standby* command includes information about the interface that is being tracked, as well as what will happen to the priority when that interface goes down:

```
Router1#show standby
FastEthernet0/1 - Group 1
  Local state is Active, priority 120, may preempt
  Hellotime 3 sec, holdtime 10 sec
  Next hello sent in 0.564
  Virtual IP address is 172.22.1.1 configured
  Active router is local
  Standby router is 172.22.1.2 expires in 9.848
  Virtual mac address is 0000.0c07.ac01
  5 state changes, last state change 12:47:08
  Priority tracking 1 interface, 1 up:
    Interface                   Decrement  State
    Serial0/0                      20      Up
Router1#
```

When this interface goes down, causing an HSRP priority change, the router will send several messages to the log buffer:

```
Jun 24 23:24:58: %STANDBY-6-STATECHANGE: FastEthernet0/1 Group 1 state Active -> Speal
Jun 24 23:25:00: %LINK-3-UPDOWN: Interface Serial0/0, changed state to down
Jun 24 23:25:01: %LINEPROTO-5-UPDOWN: Line protocol on Interface Serial0/0, changed
state to down
```

The HSRP change happens so quickly that the message actually precedes the serial interface change. This is because the serial interface doesn't send its message immediately when it loses control signals, but the HSRP change does react immediately. Upon repairing the serial interface problem, the router will send several more messages to the log:

```
Jun 24 23:25:07: %STANDBY-6-STATECHANGE: FastEthernet0/1 Group 1 state Standby ->
Active
Jun 24 23:25:08: %LINK-3-UPDOWN: Interface Serial0/0, changed state to up
Jun 24 23:25:09: %LINEPROTO-5-UPDOWN: Line protocol on Interface Serial0/0, changed
state to up
```

Again, the serial interface takes a few seconds to react, but the HSRP change is immediate. This underscores the need for the *preempt delay* command discussed in Recipe 22.2.

The *standby track* command has one other interesting application on routers that run IOS level 12.2(8)T and higher. In these versions, you can use the *keepalive* command with GRE tunnels, as discussed in Chapter 12 . With this option, GRE tunnels will mimic the behavior of physical interfaces and go into a *down* state if the far end of the tunnel becomes unavailable. This means that you can use *standby track* on a tunnel interface, which in turn means that you can now make your HSRP priority change in response to problems elsewhere in the network.

There were two important bugs with HSRP interface tracking prior to IOS level 12.1. The first happens when you track multiple interfaces. If you do not explicitly configure the priority decrement, the router will only drop the priority by a total of 10 points, no matter how many tracked interfaces fail. The second is that if the tracked interface is down at boot time and remains down, HSRP treats it as if it were up. Both of these bugs were fixed in IOS level 12.1.

22.3.4 See Also

Recipe 22.2 ; Chapter 12

Top

Recipe 22.4 Load Balancing with HSRP

22.4.1 Problem

You want to load balance your traffic between two (or more) HSRP routers.

22.4.2 Solution

You can configure HSRP so that both routers are always in use if they are available. This allows you to use your network resources more efficiently, but it is slightly more complicated to configure.

Configure the first router as follows, with two HSRP groups:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#interface FastEthernet0/1
Router1(config-if)#ip address 172.22.1.3 255.255.255.0
Router1(config-if)#standby 1 ip 172.22.1.1
Router1(config-if)#standby 1 priority 120
Router1(config-if)#standby 1 preempt
Router1(config-if)#standby 2 ip 172.22.1.2
Router1(config-if)#standby 2 priority 110
Router1(config-if)#standby 2 preempt
Router1(config-if)#end
Router1#
```

Create the same two HSRP groups on the second router, but change the priority levels from those of the first router so that Router1 is active for group 1 and Router2 is active for group 2:

```
Router2#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router2(config)#interface FastEthernet1/0
Router2(config-if)#ip address 172.22.1.4 255.255.255.0
Router2(config-if)#standby 1 ip 172.22.1.1
Router2(config-if)#standby 1 priority 110
Router2(config-if)#standby 1 preempt
Router2(config-if)#standby 2 ip 172.22.1.2
Router2(config-if)#standby 2 priority 120
Router2(config-if)#standby 2 preempt
Router2(config-if)#end
Router2#
```

This ensures that both routers back up one another simultaneously. You must then configure half of

your end devices on this segment to use the address `172.22.1.1` for their default gateway, while the other half would use `172.22.1.2`.

22.4.3 Discussion

By default, when you use HSRP on a LAN segment, all of the traffic goes through whichever router is currently active. This means that the second router and its links are generally idle. If this is a remote site and both routers have WAN links, then you will need to pay for an expensive WAN connection that is almost always unused. So this recipe shows you a way to use both routers.

This method affects only the outgoing traffic from the workstations to the routers and out to the WAN. If you also want to balance the traffic going from the WAN to the LAN, you will need to look at your routing protocol, which determines which WAN connection is the best path to this LAN segment.

The recipe is actually very simple—it simply creates two separate HSRP groups on the same segment. When everything is working normally, Router1 is the active router for one of the groups and Router2 is active for the other. Then, if either of these routers fails, the other takes over and becomes the active router for both groups.

This feature uses Multigroup HSRP (MHSRP). Not all routers support MHSRP. In particular, it does not work on Cisco 1600, 2500, 4000, or 5200/5300 devices. For Token Ring LANs, you can use MHSRP, but there are only three available HSRP groups for Token Rings. Other LAN media such as Ethernet, FDDI, ATM, 802.10, EtherChannel, and various VLAN encapsulations (including LANE, ISL, and 802.1Q) will support 256 groups. Note that you can actually configure more HSRP groups for Token Ring if you use the *use-bia* option, but every group will use the same MAC address. We discuss this option and its benefits and restrictions in Recipe 22.7.

Once you have configured the routers this way so that they both back one another up, you need to configure the end devices. Half of these devices need to have a default gateway address of `172.22.1.1`, and the other half must use `172.22.1.2`. Deciding which devices use which address is the key to balancing the load between your routers. If you configure all of your busiest devices to use the same address (and consequently, the same router), then you won't have a very well-balanced network load. This is also where the administration starts to become a little bit more complicated, because you must decide which gateway each new device will use.

Of course, if you have a situation in which both routers support two or more LAN segments, you could simply make one router primary for one segment and the other one primary for the other segment (instead of configuring two HSRP groups on the same interface). This is considerably simpler to administer and works well in larger networks.

The *show standby* command output includes information about both groups. For the first router in the example you get the following output:

```
Router1#show standby
FastEthernet0/1 - Group 1
  Local state is Active, priority 120 , may preempt
  Hellotime 3 sec, holdtime 10 sec
  Next hello sent in 1.184
  Virtual IP address is 172.22.1.1  configured
  Active router is local
  Standby router is 172.22.1.4 expires in 9.164
  Virtual mac address is 0000.0c07.ac01
  17 state changes, last state change 01:14:06
FastEthernet0/1 - Group 2
  Local state is Standby, priority 110 , may preempt
  Hellotime 3 sec, holdtime 10 sec
  Next hello sent in 2.394
  Virtual IP address is 172.22.1.2  configured
  Active router is 172.22.1.4, priority 120 expires in 8.892
  Standby router is local
  4 state changes, last state change 00:32:22
Router1#
```

You can see that this router is active for group 1 and in standby for group 2. The same command on the second router shows the converse:

```
Router2#show standby
FastEthernet1/0 - Group 1
  Local state is Standby, priority 110 , may preempt
  Hellotime 3 sec, holdtime 10 sec
  Next hello sent in 0.274
  Virtual IP address is 172.22.1.1  configured
  Active router is 172.22.1.3, priority 120 expires in 9.312
  Standby router is local
  4 state changes, last state change 01:23:46
  IP redundancy name is "hsrp-Fa1/0-1" (default)
FastEthernet1/0 - Group 2
  Local state is Active, priority 120 , may preempt
  Hellotime 3 sec, holdtime 10 sec
  Next hello sent in 2.536
  Virtual IP address is 172.22.1.2  configured
  Active router is local
  Standby router is 172.22.1.3 expires in 8.936
  Virtual mac address is 0000.0c07.ac02
  1 state changes, last state change 01:21:49
Router2#
```

22.4.4 See Also

Recipe 22.1; Recipe 22.2; Recipe 22.7

Top

Recipe 22.5 Redirecting ICMP with HSRP

22.5.1 Problem

You want to enable ICMP redirects with HSRP.

22.5.2 Solution

In older IOS releases, when you enable HSRP on an interface, the router will automatically disable ICMP redirection. However, starting with IOS Version 12.1(3)T, Cisco has changed how ICMP redirection works with HSRP, and it is now enabled by default.

You can explicitly enable ICMP redirects on HSRP-enabled interfaces with the following commands:

```
Router2#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router2(config)#interface FastEthernet 1/0
Router2(config-if)#standby redirects enable
Router2(config-if)#end
Router2#
```

The following commands prevent the router from the sending ICMP redirects on HSRP-enabled interfaces:

```
Router2#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router2(config)#interface FastEthernet 1/0
Router2(config-if)#no ip redirects
Router2(config-if)#standby redirects disable
Router2(config-if)#end
Router2#
```

The *unknown* keyword allows you to use ICMP redirection to non-HSRP routers:

```
Router2#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router2(config)#interface FastEthernet 1/0
Router2(config-if)#standby redirects unknown
Router2(config-if)#end
Router2#
```

22.5.3 Discussion

When a router receives a packet from a LAN interface, but the route to the destination points to another router on the same LAN segment, the router will send an ICMP redirect message. This is a single packet that includes information about the better route for this destination. The router will also forward the original packet over to the other router. When the end device receives the ICMP redirect packet, it updates its own internal routing table so that all future packets for this destination use the better router.

But ICMP redirection is not usually a good idea with HSRP because it will cause the end device to update its internal routing table to use the real IP address and MAC address of one of the routers when it tries to communicate with a particular remote segment. If this router were to fail, all communication to this remote segment would stop. However, the new functionality resolves this problem by using only the virtual IP and MAC addresses if the other router is running HSRP. If the other router doesn't run HSRP, then it must use the physical addresses.

This also implies that you will never see an ICMP redirect to an HSRP router that is not in the active state, because the standby router doesn't have a virtual MAC address.

Top

Recipe 22.6 Manipulating HSRP Timers

22.6.1 Problem

You want to decrease the amount of time it takes for the backup router to take over after the primary router fails.

22.6.2 Solution

You can configure HSRP-enabled routers to recover more quickly after the primary HRSP router becomes unavailable with the *standby timers* configuration command:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#interface FastEthernet0/1
Router1(config-if)#standby 1 ip 172.22.1.1
Router1(config-if)#standby 1 priority 120
Router1(config-if)#standby 1 preempt
Router1(config-if)#standby 1 timers 1 3
Router1(config-if)#end
Router1#
```

If you change the HSRP timers on one router, you must change the timers on all of the other routers in the same group:

```
Router2#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router2(config)#interface FastEthernet1/0
Router2(config-if)#standby 1 ip 172.22.1.1
Router2(config-if)#standby 1 priority 110
Router2(config-if)#standby 1 preempt
Router2(config-if)#standby 1 timers 1 3
Router2(config-if)#end
Router2#
```

22.6.3 Discussion

By default, a router will send HSRP hello packets every 3 seconds, and a standby router will declare itself active if it doesn't hear any hello packets from the active router for 10 seconds. The command in the example changes the timers from these default values to a 1-second hello period and 3-second failover:

```
Router1(config-if)#standby 1 timers  1 3
```

With this command, the standby router can reduce the amount of outage time by 7 seconds. This might be useful in some highly mission-critical networks.

The *show standby* command output includes the timer settings so you can easily see what values the router is using:

```
Router1#show standby
FastEthernet0/1 - Group 1
  Local state is Active, priority 120, may preempt
  Hellotime 1 sec, holdtime 3 sec
  Next hello sent in 0.420
  Virtual IP address is 172.22.1.1 configured
  Active router is local
  Standby router is 172.22.1.2 expires in 2.968
  Virtual mac address is 0000.0c07.ac01
  5 state changes, last state change 12:50:25
Router1#
```

Decreasing the HSRP hello interval also increases the amount of background chatter on the network segment. But, since HSRP routers exchange these packets using IP multicast, it is relatively efficient. Even if there are many routers in the same group, each packet only appears on the network once. And, unlike broadcasts, these packets shouldn't cause problems for other devices on the segment. It is unlikely that anybody would want to increase these timers from their default values.

If you need the timers to be even faster, you can configure millisecond intervals:

```
Router1(config-if)#standby 1 timers msec  100 msec 300
```

With the *msec* keyword, Cisco routers will accept a range of 50 to 999 milliseconds for the hello interval and 200 to 3000 milliseconds for the hold timer. Without this keyword, the hello interval must be between 1 and 254 seconds, and the hold timer must be between 2 and 255 seconds.

Be extremely careful with these millisecond timings! They can cause two potential problems. If the timers are too short (for example, if you set the hello timer to 50 milliseconds on a busy 4Mbps Token Ring) you could cause congestion problems on the ring. The second problem to watch out for is using a timer that is too short for the network to reliably deliver. For example, if your LAN includes any bridges (including ATM LAN extensions) to remote sites, you may get spurious transitions just caused by random variation in latencies (jitter).

The HSRP RFC states that the hold time should be at least three times the value of the hello time and MUST be greater then the hello time. Setting your hold time to three times the hello time is a good rule of thumb. This allows the router to miss two packets due to congestion or random noise without disrupting the way the network functions.

22.6.4 See Also

RFC 2281

Top

Recipe 22.7 Using HSRP on a Token Ring Network

22.7.1 Problem

You want to configure HSRP on a Token Ring network.

22.7.2 Solution

You can use HSRP on a Token Ring LAN exactly the same as in Recipe 22.1 if the only protocol on the segment is IP. However, if you have any other protocols (particularly if the ring uses any source-route bridging), you must use a slightly different configuration:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#interface Tokenring0
Router1(config-if)#ip address 172.22.1.3
Router1(config-if)#standby ip 172.22.1.1
Router1(config-if)#standby use-bia
Router1(config-if)#standby priority 120
Router1(config-if)#standby preempt
Router1(config-if)#end
Router1#
```

The second router is similarly configured:

```
Router2#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router2(config)#interface Tokenring0
Router2(config-if)#ip address 172.22.1.2
Router2(config-if)#standby ip 172.22.1.1
Router2(config-if)#standby use-bia
Router2(config-if)#standby priority 110
Router2(config-if)#standby preempt
Router2(config-if)#end
Router2#
```

22.7.3 Discussion

As we discussed in Chapter 15, the biggest functional difference between a Token Ring LAN and an Ethernet LAN is that Token Ring bridging is usually *source-routed*, while Ethernet almost always uses *transparent bridging*. Consequently, Token Ring devices make use of a Routing Information Field

(RIF) that contains MAC address information.

This is particularly important when the two HSRP routers reside on different physical rings connected by a bridge. In this case, when HSRP changes the active router, the bridges will see the virtual MAC address suddenly jump from one ring to another, which will disrupt the RIF tables in every bridge.

So, when using HSRP in Token Ring environments, it is usually best to just use the router's Burned-In Address instead of a virtual MAC address. Then HSRP will rely on gratuitous ARP packets to update the ARP cache of every device on the ring, and tell it that the default gateway router has changed.

The key command in this configuration is the *use-bia* command, which appears on both routers:

```
Router1(config-if)#standby use-bia
```

In this case, we are using standby group 0, because we didn't specify a group number, so the virtual MAC address would have been `C0-00-00-01-00-00` if we had not changed it to the BIA. The output of a show standby command shows the address that the router is actually using:

```
Router1#show standby
TokenRing0 - Group 0
  Local state is Active, priority 120, may preempt, use bia
  Hellotime 3 holdtime 10
  Next hello sent in 00:00:02.338
  Hot standby IP address is 172.22.1.1 configured
  Active router is local
  Standby router is 172.22.1.3 expires in 00:00:09
  Standby virtual mac address is 0000.300f.2186
Router1#
```

We chose to use standby group 0 in this example because, prior to IOS level 12.0(3.4)T, you could use this feature only with group 0. In newer IOS levels, however, this feature is available for all Token Ring HSRP groups. Without this option, there are only three usable HSRP groups because there are only three distinct MAC addresses. With this option, however, you can configure up to 255 groups, all of which will use the same BIA MAC address.

The *use-bia* option can also be useful in non-Token Ring environments. Networks that use DECnet or Xerox Network Services (XNS) frequently encounter MAC address problems as well, even on Ethernet LANs. This is because devices that run both protocols will see two different MAC addresses for the same destination device, which causes confusion. We also recommend using the Burned-In Address in these types of environments.

You should be aware of two important limitations to this command. First, and most important, when you use this option, HSRP must rely on gratuitous ARP to tell end devices that the MAC address of the default gateway has changed. However, some devices do not handle gratuitous ARP packets well. In some implementations, a device will update its ARP cache only if it receives an ARP packet in response to a specific ARP request.

The second limitation is that this option completely breaks Proxy ARP. This is because the end devices believe that the remote IP address is associated with the MAC address of the primary HSRP router. However, the backup router has no way of knowing which remote IP addresses the primary router has sent out Proxy ARP messages for. Therefore, it can't update these entries to tell the end devices to use the new MAC address instead.

We know of no workarounds for either of these problems, so please use caution when implementing this feature.

22.7.4 See Also

Recipe 22.1, Recipe 22.2; Chapter 15

Top

Recipe 22.8 HSRP SNMP Support

22.8.1 Problem

You want to enable HSRP SNMP traps.

22.8.2 Solution

Cisco has developed an HSRP SNMP MIB to help manage routers using this feature. You can configure your router to send an SNMP trap every time the routers make an HSRP state change:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#snmp-server enable traps hsrp
Router1(config)#snmp-server host 172.25.1.1 ORATRAP
Router1(config)#end
Router1#
```

22.8.3 Discussion

When a router changes its HSRP state, it usually indicates that some sort of network problem has occurred. If you are tracking interfaces (as described in Recipe 22.3), the problem may be that the WAN interface on the primary router has failed. In this case, you will almost certainly also receive a link-down trap from the primary router, as well as a trap indicating that the primary has become inactive and the secondary router has become active.

In some cases, it can be extremely useful to receive these traps. For example, the primary router's power supplies may have suddenly failed. In this case, the first indication that there has been a problem may come when your network management system next tries to poll the primary router. Clearly, there are cases where you want to know about the change sooner than that. So having the secondary router send a trap to indicate the change of HSRP state may be the fastest way to get this information.

These traps can also be useful if there is something wrong with your HSRP setup that causes the two routers to continually flip back and forth between active and standby states. As long as the traffic continues to flow, you may not otherwise know that there is a problem. This could happen, for example, if you set the HSRP timers improperly.

Cisco's HSRP MIB also allows you to use SNMP to query the routers for their current HSRP state information. Refer to Chapter 17 for more information about using SNMP.

22.8.4 See Also

Recipe 22.3; Recipe 22.6; Chapter 17

Top

Recipe 22.9 Increasing HSRP Security

22.9.1 Problem

You want to increase the security of HSRP between two (or more) routers.

22.9.2 Solution

You can configure HSRP to use password authentication with the following commands:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#interface FastEthernet 0/1
Router1(config-if)#standby 1 ip 172.22.1.1
Router1(config-if)#standby 1 priority 120
Router1(config-if)#standby 1 authentication OREILLY
Router1(config-if)#end
Router1#
```

You must configure the same authentication password on all routers within the same HSRP group, or the conflicts will prevent HSRP from working:

```
Router2#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router2(config)#interface FastEthernet 1/0
Router2(config-if)#standby 1 ip 172.22.1.1
Router2(config-if)#standby 1 priority 110
Router2(config-if)#standby 1 authentication OREILLY
Router2(config-if)#end
Router2#
```

To prevent any other routers from becoming active, set the primary router's priority to the highest possible value, 255:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#interface FastEthernet 0/1
Router1(config-if)#standby 1 ip 172.22.1.1
Router1(config-if)#standby 1 priority 255
Router1(config-if)#end
Router1#
```

Then you can configure the standby router to use a slightly lower priority number:

```
Router2#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router2(config)#interface FastEthernet 1/0
Router2(config-if)#standby 1 ip 172.22.1.1
Router2(config-if)#standby 1 priority 254
Router2(config-if)#end
Router2#
```

This will help to ensure that no other routers that might be on this segment can take over because of an HSRP coup.

22.9.3 Discussion

HSRP is not a terribly secure protocol, even with the precautions shown in this recipe. This is usually not a problem, however, because most network engineers only use it on internal, trusted LAN segments.

HSRP has two main security related problems. The first is simply caused by incorrect router configuration. It is possible to cause serious routing problems if more than one router is active, if no routers are active on a segment, or if the wrong router becomes active. The second potential security problem is that a hostile user can configure a device, such as another Cisco router, to take over as the HSRP active router. They might use this to capture and examine packets that they would not otherwise see in a switched LAN, route packets to a different network, or cause a simple DoS attack. However, because HSRP uses the locally scoped multicast address `224.0.0.2`, with a TTL of 1, it is extremely unlikely that anybody could launch an effective HSRP attack if they were not physically connected to this LAN segment.

You can use HSRP authentication to help prevent misconfigured routers from becoming active on a production LAN. The routers send the authentication password through the network in unencrypted clear-text using IP multicast, so it is relatively easy for any device on the LAN segment to determine this password.

The following is an HSRP hello packet that was captured using Ethereal:

```
Cisco Hot Standby Router Protocol
    Version: 0
    Op Code: Hello (0)
    State: Active (16)
     Hellotime: Default (3)
     Holdtime: Default (10)
    Priority: 120
    Group: 1
    Reserved: 0
    Authentication Data: Non-Default (OREILLY)
     Virtual IP Address: 172.22.1.1
```

Note that all of the important HSRP information, including timers, priorities, the group number, and even the virtual IP address are readily available to anybody who captures HSRP packets on her local LAN segment. This illustrates both how insecure HSRP is and how easy it would be to create a false HSRP device to maliciously disrupt LAN communication.

The biggest problem with HSRP authentication—and the reason why you may decide not to use it—appears when the passwords on two routers in the same group do not agree. The two routers have no particular way of knowing which password is correct, so they both assume that the other is wrong. This can cause both routers to become active, which is not at all desirable. This feature is obviously not a very good way to prevent a malicious user from taking over control of the gateway.

If HSRP routers in the same group are configured with different authentication passwords, you will see the following messages in their logs:

```
Jun 25 11:00:15: %STANDBY-3-BADAUTH: Bad authentication from 172.22.1.4, group 1,
remote state Standby
```

Cisco's intention is that this feature should be used to prevent other routers from learning HSRP parameters, such as the virtual IP address and timer information. However, we don't generally advise using it to address real security requirements.

You can use the *show standby* command to verify your HSRP authentication information:

```
Router1#show standby
FastEthernet0/1 - Group 1
  Local state is Active, priority 120, may preempt
  Hellotime 1 sec, holdtime 3 sec
  Next hello sent in 0.754
  Virtual IP address is 172.22.1.1 configured
  Active router is local
  Standby router is 172.22.1.2 expires in 2.824
  Virtual mac address is 0000.0c07.ac01
  Authentication text "OREILLY"
  5 state changes, last state change 12:56:36
Router1#
```

The last example in the solution section to this recipe shows how to configure your router to avoid another type of attack. A rogue user could configure a router with a higher priority than the current active router. This would cause an HSRP coup, and the rogue router would be able to take over as the active router. This illegitimate router could then freely manipulate routing for this segment.

You can partially guard against this scenario by setting your primary router to the highest possible priority level. This should prevent a rogue router from forcing a priority election. However, recall that when two routers have the same HSRP priority, the one with the higher physical IP address will win the election. So, if you have good reason to be concerned about this type of attack, we recommend using the highest possible IP addresses on the segment for your physical IP addresses as well as the highest possible priorities.

The output of this *show standby* command highlights the priority value:

```
Router1#show standby
FastEthernet0/1 - Group 1
  Local state is Active, priority 255, may preempt
  Hellotime 1 sec, holdtime 3 sec
  Next hello sent in 0.436
  Virtual IP address is 172.22.1.1 configured
  Active router is local
  Standby router is 172.22.1.2 expires in 2.508
  Virtual mac address is 0000.0c07.ac01
  Authentication text "OREILLY"
  5 state changes, last state change 13:00:48
Router1#
```

You can further improve your HSRP security by using IPSec to encrypt HSRP packets. This will ensure that rogue users can't affect how HSRP functions on your LAN segment. This is a secure method, but full coverage is outside the scope of this chapter. This type of hardening isn't usually required on enterprise networks; if you have internal users that are this hostile, securing HSRP is probably the least of your worries. However, if you need HSRP on a public network (such as an Internet connection), it might be worth considering. Note, however, that this is not necessary even in most Internet connection situations because the other routers on the segment will almost certainly be ISPs (if you are a customer) or a customer (if you are an ISP). These organizations will have a vested interest in a stable network, so they are unlikely to want to break things. The only place where you are likely to need extra security is on LAN segments that have many routers to many different networks, none of which you control. In this case, you could configure an IPSec encrypted tunnel between the LAN interfaces of the two routers, and use a route map to force HSRP packets through the tunnel. Refer to Chapter 12 for more information on IPSec.

22.9.4 See Also

Recipe 22.1; Chapter 12

Top

Recipe 22.10 Showing HSRP State Information

22.10.1 Problem

You want to see current HSRP information, such as which router is primary.

22.10.2 Solution

To view the HSRP information, use the following EXEC command:

```
Router2#show standby
```

You can view the HSRP information for a specific interface with the following EXEC command:

```
Router2#show standby FastEthernet 1/0
```

Use the keyword *brief* to show an overview of HSRP information:

```
Router2#show standby brief
```

22.10.3 Discussion

The basic *show standby* command without any additional keywords displays all of the HSRP information for all groups and all interfaces on the router:

```
Router2#show standby
FastEthernet1/0 - Group 1
  Local state is Standby, priority 110, may preempt
  Hellotime 1 sec, holdtime 3 sec
  Next hello sent in 0.536
  Virtual IP address is 172.22.1.1 configured
  Active router is 172.22.1.3, priority 255 expires in 2.380
  Standby router is local
  Authentication text "OREILLY"
  1 state changes, last state change 15:43:34
  IP redundancy name is "hsrp-Fa1/0-1" (default)
Router2#
```

If your router runs HSRP on several interfaces, you might want to just look at the HSRP status for a particular interface:

```
Router2#show standby FastEthernet 1/0
```

```
FastEthernet1/0 - Group 1
  Local state is Standby, priority 110, may preempt
  Hellotime 1 sec, holdtime 3 sec
  Next hello sent in 0.036
  Virtual IP address is 172.22.1.1 configured
  Active router is 172.22.1.3, priority 255 expires in 2.796
  Standby router is local
  Authentication text "OREILLY"
  1 state changes, last state change 15:47:44
  IP redundancy name is "hsrp-Fa1/0-1" (default)
Router2#
```

The *show standby brief* command is particularly useful when you have many HSRP interfaces or groups. This command presents all of the most important information for each group on a single line:

```
Router2#show standby brief
                     P indicates configured to preempt.
                     |
Interface   Grp  Prio P State    Active addr     Standby addr    Group addr
Fa1/0       1    110  P Standby  172.22.1.3      local           172.22.1.1
Fa1/0       2    120  P Active   local           172.22.1.3      172.22.1.2
Router2#
```

Top

Recipe 22.11 Debugging HSRP

22.11.1 Problem

You want to debug an HSRP problem.

22.11.2 Solution

To debug all HSRP error events, use the following command:

```
Router2#debug standby errors
```

The *events* keyword will display information about HSRP events:

```
Router2#debug standby events
```

With the *packets* keyword, you can look at the contents of all HSRP packets:

```
Router2#debug standby packets
```

You can use the *terse* keyword to see a short form of all HSRP errors, events, and packets:

```
Router2#debug standby terse
```

22.11.3 Discussion

HSRP is not a very complex protocol, and it is relatively simple to configure, so network engineers generally don't find that they need the sophisticated debugging tools that are available with other protocols. Consequently, HSRP debugging facilities were relatively limited until IOS level 12.1(0.2), when the enhanced debugging described here was introduced. However, these features can be useful when you are faced with strange HSRP problems such as general instability or multiple active routers.

We don't recommend starting with a packet-level debug for anything because it can easily overwhelm the router. In the case of HSRP, which should only send a hello packet every three seconds by default, this shouldn't be quite as dangerous as for many other protocols.

The *debug standby terse* command is probably the most useful option because it gives a short form output of all HSRP errors, events, and packets:

```
Router1#debug standby terse
HSRP:
```

```
  HSRP Errors debugging is on
  HSRP Events debugging is on
    (protocol, redundancy, track)
  HSRP Packets debugging is on
    (Coup, Resign)
Router1#
```

From here, if you see a coup problem (for example), you might want to turn off the *terse* option and replace it with a full packet-level debug. Conversely, if you see that there is an issue with interface tracking, you might want to replace the *terse* option with the *event* option to get greater detail.

Top

Chapter 23. IP Multicast

Top

Introduction

Multicast routing differs from unicast routing in several ways. The most important differences are in the ways that multicast routers use source and destination addresses. A multicast packet is addressed to a special IP address representing a group of devices that can be scattered anywhere throughout a network. Since the destinations can be anywhere, the only reliable way to eliminate loops in multicast routing is to look at the reverse path back to the source. So, while unicast routing cares about where the packet is going, multicast routing also needs to know where it came from.

For this reason, multicast routing protocols such as Protocol Independent Multicast (PIM) always work with the source address and destination group simultaneously. The usual notation for a multicast route is *(Source, Group)*, as opposed to the unicast case, in which routes are defined by the destination address alone. We have already mentioned that this is necessary for avoiding loops, but the router also needs to keep track of both source and group addresses in each multicast routing table entry because there could be several sources for the same group.

For example, in Chapter 14 we discussed how a central device can use NTP to send time synchronization information as a multicast router. We also explained why it was important to have more than one NTP server. So, even in a simple multicast example like this it is quite likely that the routers will need to forward packets to the same set of end devices from two sources that may be on different network segments. The group address alone doesn't tell you enough about how to forward packets belonging to this group.

When you look at the multicast routing table with the *show ip mroute* command, you will see not only *(Source, Group)* pairs such as `(192.168.15.35, 239.5.5.55)`, but also pairs that look like `(*, 239.5.5.55)`. This means that the source is unspecified. Cisco routers organize their multicast routing tables with a parent *(*, Group)* for each group, and any number of *(Source, Group)* pairs under it. If there is a *(*, Group)*, but no *(Source, Group)* entries for a group, then that just means that the router knows of group members but doesn't yet know where to expect this multicast traffic from.

Each of these *(Source, Group)* entries represents a Shortest Path Tree (SPT) that leads to the source of the multicast traffic. In sparse mode multicast routing, the root of the tree could actually be a central Rendezvous Point (RP) router, rather than the actual traffic source. Because each router must know about the path back to the source or RP, the term Reverse Path Forwarding (RPF) is often used to describe the process of building the SPT.

Two important elements are required for a multicast network to work. The first we've already mentioned: you need a way to route multicast packets from the source to all of the various destinations in the group. The other critical element is that the multicast network has to provide a way for end

devices to subscribe to a multicast group so that they can receive the data. The network uses the Internet Group Management Protocol (IGMP) to manage group subscriptions.

IGMP and CGMP

IGMP functions mainly at Layer 3. Individual end devices use IGMP to announce that they wish to join a particular multicast group. The IGMP request is picked up by a router that attempts to fulfill the request by forwarding the multicast data stream to the network containing this device. The IGMP protocol is in its second version, which is defined in RFC 2236. A third version is currently in the draft stages.

What IGMP does is relatively simple in concept. It provides a method for end devices to join and leave multicast groups. Here is the output of *tcpdump* showing the device `192.168.1.104` joining the group `239.5.5.55`:

```
17:10:16.397055 192.168.1.104 > 239.5.5.55: igmp nreport 239.5.5.55 (DF) [ttl 1]
17:10:19.276998 192.168.1.104 > 239.5.5.55: igmp nreport 239.5.5.55 (DF) [ttl 1]
17:10:21.027002 192.168.1.104 > 239.5.5.55: igmp nreport 239.5.5.55 (DF) [ttl 1]
```

Note that the device sends three IGMP packets stating its membership to make sure that it is heard. The router receives the request to join this group and sets a timer to count down for three minutes. As long as some device reasserts its membership with IGMP within this period, the group will remain in the router's multicast routing table. If all of the group members leave, or if they all simply stop sending IGMP updates for more than three minutes, the router will remove this group from its tables to save memory.

When the device wants to stop receiving a multicast group, it sends a single IGMP leave packet. The router immediately reacts by sending a query to this segment to find out if there are still any other members left in this group. It tries twice before deciding to stop sending traffic for this group to this network segment:

```
17:16:17.934667 192.168.1.104 > ALL-ROUTERS.MCAST.NET: igmp leave 239.5.5.55 (DF)
[ttl 1]
17:16:17.937715 192.168.1.1 > 239.5.5.55: igmp query [gaddr 239.5.5.55] [tos 0xc0]
[ttl 1]
17:16:19.050430 192.168.1.1 > 239.5.5.55: igmp query [gaddr 239.5.5.55] [tos 0xc0]
[ttl 1]
```

The important changes to the protocol between Versions 1 and 2 of IGMP have to do with determining when all of the members of a group in a particular network have left. The most important addition to Version 3 is the ability to specify and filter multicast sources. So a device may specify that it is interested in receiving multicast messages from one source, but not from another—even though both sources may be sending to the same group.

Although it is not yet fully standard and few end devices support IGMP Version 3, Cisco has already adopted these extensions. You lose nothing by implementing them now because the IGMP protocol is

fully backward compatible.

In a switched Ethernet LAN (including 100Mbps, 1000Mbps and higher speed variants), there is an additional benefit to multicast transmission. If the switches are multicast aware, they can forward packets with a particular group address to only those devices that are members of this group. So it is not necessary to flood the entire VLAN with multicast packets just because one device is a multicast group member. Naturally, this means that the switch must be able to read and use Layer 3 information, so this sort of functionality is not available on all Ethernet switches.

Many multicast-aware switches use *IGMP snooping* to read IGMP packets from devices as they join and leave particular groups. This sounds like a perfect and simple solution, but, in practice, it can be very complex to implement in the switch. The first problem is that there are several special cases that are difficult to manage. For example, things become quite complex when you have several multicast routers on a segment, or when there are complicated trunk topologies or connections to workgroup hubs. Another important problem with IGMP snooping is that the switch must read the contents of all multicast packets passing through it so that it won't miss any IGMP Join or Leave messages. In effect, the switch acts as if it were a member of every multicast group. If there is a heavy multicast application such as a multi-media application, this can cause serious CPU overhead on the switch.

Cisco has developed a proprietary protocol called Cisco Group Management Protocol (CGMP) to deal with these problems. CGMP is implemented on all Cisco routers and most new switches, even those without Layer 3 capabilities. It is a relatively simple protocol that allows the router to do most of the hard work for the switch. When a device on the LAN segment joins a multicast group by sending an IGMP Join message, the switch simply passes the IGMP packet through to the router as it would with any other packet. The router then sends a CGMP packet to the switch to let it know the MAC addresses of the device and the group. Similarly, when a device leaves a group, the router uses CGMP to tell the switch to stop forwarding this particular multicast group to this device. In this way the router, which has to keep track of this information anyway, can simply tell the switch what to do.

Unfortunately, CGMP doesn't solve all of the problems inherent in the IGMP model. Specifically, a device doesn't need to send an IGMP *Leave* message when it is no longer interested in receiving packets for that group. If the last group member leaves without sending the appropriate IGMP *Leave* message, the router will still think that there are devices in the group. It will continue to forward multicast packets to the segment until a timer expires. The router will eventually poll the LAN segment to see if any devices are still interested in receiving this group. If it gets no response, it will finally stop sending the multicast data stream. However, most implementations of IGMP Version 2 do send explicit *Leave* messages unless the end devices crash or terminate improperly. In any case it is usually better to have a device receive multicast data it didn't subscribe to than to lose the data. The only time when this isn't true is when the multicast data stream consumes too much bandwidth and starts to cause congestion for normal unicast traffic, or when processing the unnecessary multicast traffic causes CPU problems on the end devices.

Switches running newer versions of CGMP include a particularly nice feature called Local Leave

Processing. With it, the switches are able to intercept IGMP Leave messages from devices and process them internally. If there are other group members elsewhere on the switch, it can simply stop sending data from this group to the device that no longer wishes to be a member. Then, when the last group member leaves the group, the switch will send a global IGMP *Leave* packet to the router to tell it to stop sending this multicast group.

Multicast Routing Protocols

There are two general types multicast routing protocols, called *dense* and *sparse* mode. Dense mode means that every multicast router receives every multicast packet unless and until it explicitly says that it doesn't want it. As we will discuss shortly, this applies to each group and each interface separately. Sparse mode, on the other hand, means (loosely) that no router will receive a multicast group unless it explicitly requests it. It is important to note that end devices, whether multicast servers or group members, are completely unaware of which mode their network uses, or even which multicast routing protocol. Indeed it is possible to run a network where the routers use a combination of these modes.

There are many examples of dense-mode protocols, such as Protocol Independent Multicast-Dense Mode (PIM-DM), Distance Vector Multicast Routing Protocol (DVMRP), and Multicast Open Shortest Path First (MOSPF). There are fewer sparse-mode protocols, with the best examples being Protocol Independent Multicast-Sparse Mode (PIM-SM) and Core Based Trees (CBT).

> Not all of these protocols are available in Cisco routers. Like most vendors, Cisco implements PIM-DM and PIM-SM, as well as MBGP. But Cisco does not implement MOSPF or CBT, and has a limited version of DVMRP.

There are two other general categories of multicast routing protocols: protocol dependent and protocol independent. The difference has to do with the interaction with an underlying routing protocol, not with the ability to handle non-IP multicast traffic. All of the multicast protocols mentioned in this book are specific to IP multicast communications.

For example, MOSPF is protocol dependent because it relies on OSPF, and uses a special OSPF LSA type to carry information about multicast routing. PIM and CBT, on the other hand, both use the multicast traffic itself, along with the standard unicast IP routing table and IGMP requests to build the multicast forwarding trees. Since they don't care how the router got its unicast IP routing table, they are called protocol independent.

For the network engineer these distinctions are quite important, since they affect flexibility, reliability, and network performance. In general, if you have a large network (particularly one with bandwidth-constrained WAN links) and the multicast sources and destinations can be more or less anywhere through your network, you should use a protocol independent sparse mode multicast routing protocol. Even if you're not sure that this really describes your network, it is generally safer and easier to lean in this direction.

The PIM protocols, in particular PIM-SM, are generally the best choices for implementing new multicast networks. In the past there were problems with interoperability in multivendor networks as different router manufacturers implemented different sets of multicast routing protocols. Since DVMRP was the first widely implemented multicast routing protocol, the rule of thumb used to be that DVMRP was the best way to allow communication between groups of routers from different vendors. However, a quick survey of protocols supported by major router vendors shows that almost all of them now support PIM and DVMRP.

PIM-DM and PIM-SM

PIM-DM and PIM-SM have several important similarities, as well as important differences. Let's look schematically at how each builds and maintains its multicast forwarding trees to explain how they work. Note that this is not intended to be a rigorous explanation of the protocols. Instead, we just want to give you a good basic understanding of what they do and how they do it. For more detailed information, refer to the standards documents, particularly RFCs 2362 and 2715.

Suppose a device wants to join a group, G. The first thing it does is to send an IGMP Join message to its local router. If this is the first group member, the router creates an entry in its multicast forwarding table for (*,*G*). This says to forward to this interface all multicast packets addressed to group G from any source. At this point, if the router receives any packets for this group, it knows at least one place to which to forward them.

In PIM-DM, the router will create the group and wait for packets. It will also send a *Join* request to each of its PIM neighbors to find out if they have this group. If it receives multicast packets for a group that it doesn't care about, then the router will send *Prune* messages back to where they came from, to ask to be removed from the forwarding tree for this group. This is commonly called a "flood and prune" model, which is common to all dense mode multicast protocols.

If this router uses PIM-SM, however, it will attempt to join a multicast tree rooted at the RP. An RP is a router somewhere in the network that acts as a central distribution point for one or more multicast groups. We will discuss how the other routers come to know about the RP in Recipe 23.2 and Recipe 23.3, but for now we'll assume that they know how to find it. When the last-hop router receives an IGMP message from a device asking to join a group, it has to go looking for that group. The best place to start looking is the RP.

So, the last-hop router looks at its unicast routing table to figure out which of its neighboring routers is the best path to the RP, and it sends it an explicit PIM-SM Join message for this group. If the neighboring router is already receiving this group, then the problem is solved and the data starts to flow. Otherwise, this neighbor must send another Join to the next-hop router in the direction of the RP, and so on until a multicast-forwarding tree is created with its root at the RP.

The upstream router will automatically prune the branches of this multicast tree if they don't receive

another explicit Join within the three-minute timeout period. So, by default, the routers all refresh the tree with a new Join for every active group once per minute. This creates and maintains a stable tree rooted at the RP and extending to all group members in the network that remains active even if there is no multicast traffic being forwarded.

The only remaining piece of the puzzle is how the packets get from the sender to the RP. When the source device sends its first packet, the first-hop router receives it normally as it would any other packet. This first-hop router has already learned where the RP is. When it receives a multicast packet from a new source, the router must register this source with the RP. The router encapsulates the multicast packet in a PIM-SM registration packet, which it sends by unicast to the RP. The RP then removes the encapsulation and forwards the packet down the tree. The RP also sends an explicit PIM-SM Join message toward the source. The Join message links up a tree from the RP upstream to the source and downstream to the pre-existing tree containing the group members. Once the tree is built, there is no need for the first-hop router to continue encapsulating multicast packets to send them to the RP. So the first-hop router can revert to normal multicast forwarding instead, knowing that the RP is somewhere downstream on the SPT.

Finally, once there is a tree connecting the ultimate source with all of the group members, there is no more need for the RP. So the last-hop routers start to send PIM-SM Join messages to create a new tree that is centered on the source rather than the RP. This is actually controlled by a minimum traffic flow threshold. PIM-SM starts to build the new tree rooted at the source only if the amount of traffic coming down the tree for this group exceeds this threshold. This threshold traffic flow rate is zero by default on Cisco routers, but we will show how to adjust it in Recipe 23.4

For LAN segments that have more than one multicast router, PIM also includes the concept of a Designated Router (DR), which preferentially handles multicast forwarding for this segment. The DR ensures that each multicast application packet appears on the LAN segment once and only once. This is similar to the OSPF Designated Router concept that we discussed in Chapter 8, but PIM's election process is much simpler. The PIM DR is just the router whose interface on this segment has the higher IP address. Whenever there are two multicast routers on the same segment, they learn about one another through PIM, and periodically send special multicast "Hello" packets to one another. This ensures that, if the current DR device becomes unavailable, the next candidate can take over this function. Since the backup device resides on the same LAN segment, it is able to monitor IGMP, so it always knows which groups it will need to forward if it needs to become the DR.

DVMRP

Distance Vector Multicast Routing Protocol (DVMRP) is defined in RFC 1075, and was the first widely implemented multicast routing protocol. This protocol is similar to RIP in many ways. There are a few important differences, though. The maximum diameter of a RIP network is 16 hops, as we mentioned in Chapter 6. DVMRP has a maximum metric of 32, which drastically improves its flexibility. It's not hard to find a network with a diameter greater than 16 hops, but a 32-hop diameter is sufficient for

most real-world corporate networks. It is not sufficient for the public Internet, but that is why multiprotocol extensions to the Border Gateway Protocol, sometimes also called Multicast Border Gateway Protocol (MBGP), were invented.

DVMRP is often a good choice for allowing routers from different manufacturers to exchange multicast routing information. It is a dense mode protocol, however, so it is generally less efficient with network resources. We recommend using DVMRP primarily as a mechanism for exchanging multicast routing information with older non-Cisco devices. In recent years, PIM has become the popular choice for multicast routing among most large router vendors, though, so DVMRP's niche is now mostly in interconnecting with existing non-Cisco multicast networks.

In many ways, DVMRP functions in a similar way to PIM-DM. It uses a dense-mode strategy that forces all routers to prune themselves from any multicast trees that they don't require. And it also uses the unicast routing table to determine the shortest path back to the source device. The main difference, however, is that DVMRP includes its own internal unicast routing protocol that it uses to help make decisions about the best SPT.

DVMRP uses an algorithm called Truncated Reverse Path Broadcasting (TRPB) to allow every router in the network to determine where it is relative to the multicast source, and to calculate the optimal SPT back to the source. Because DVMRP uses its own internal unicast routing protocol, it is not considered protocol independent.

You must take special measures to force DVMRP to follow the standard unicast routing table and make it protocol independent. Of course, this would break one of the main reasons for using DVMRP in the first place. Because it maintains its own routing tables, DVMRP is able to work in networks where the multicast and unicast topologies are different. This is not uncommon in cases where parts of the unicast network don't support multicast routing, or where traffic engineering leads you to put multicast traffic through different network links.

In fact, Cisco routers do not provide a full DVMRP implementation. They can take part in discovering and exchanging routing information with DVMRP neighbors. But the actual multicast routing is done using PIM while referring to the DVMRP routing tables.

MOSPF

Multicast Open Shortest Path First (MOSPF) is not really a separate protocol, but rather is a set of extensions to the popular Open Shortest Path First (OSPF) unicast routing protocol. OSPF is described in more detail in Chapter 8. To allow OSPF to carry multicast routing information, RFC 1584 added a new Link State Advertisement (LSA) type called Type 6, or simply the MOSPF LSA.

Cisco routers do not support MOSPF, so we will not discuss this protocol in any detail except to point out that Cisco routers will generate log error messages whenever they encounter Type 6 OSPF LSAs. Recipe 23.7 shows how to configure the router to ignore these packets.

The biggest advantage to MOSPF is that it is tightly integrated with OSPF, which can simplify network administration. Furthermore, because it uses the same Link State algorithm as OSPF, every router in the network can independently deduce the best path back to the source.

However, it is a dense-mode protocol, consequently less efficient with network resources, and requires OSPF to work. This is almost certainly why Cisco has chosen not to implement it.

MBGP

Multicast Border Gateway Protocol (MBGP) is based on a small set of extensions to BGP defined in RFC 2858 to allow the exchange of any routable protocol information between ASes. It does this by simply introducing two new attributes to the BGP protocol: Multiprotocol Reachable Network Layer Routing Information (*MP_REACH_NLRI*) and Multiprotocol Unreachable Network Layer Routing Information (*MP_UNREACH_NLRI*), which are used to carry information about reachable and unreachable networks.

It's important to understand that MBGP is not really a multicast routing protocol in the same sense as PIM or DVMRP. It doesn't understand or have the ability to *Join* or *Prune* SPTs. It doesn't include any functionality for dealing with Rendezvous Points. All it does is forward information about multicast groups and sources, and make this information available to other multicast routing protocols. It needs another protocol to do all of the other work of joining and pruning multicast distribution trees. The two protocols most commonly used for this are PIM and DVMRP.

Top

Recipe 23.1 Configuring Basic Multicast Functionality with PIM-DM

23.1.1 Problem

You want to pass multicast traffic through the router.

23.1.2 Solution

In a small network with few routers and relatively light multicast application bandwidth requirements, the easiest way to implement multicast routing is to use PIM-DM. This example shows the configurations for two routers that are connected through a serial connection, both with FastEthernet interfaces to represent the LAN connections. It is important to enable multicast routing on all interfaces that connect to other multicast-enabled routers or to multicast user or server segments. The first router looks like this:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#ip multicast-routing
Router1(config)#interface FastEthernet0/0
Router1(config-if)#ip address 192.168.1.1 255.255.255.0
Router1(config-if)#ip pim dense-mode
Router1(config-if)#exit
Router1(config)#interface Serial1/0
Router1(config-if)#ip address 192.168.2.5 255.255.255.252
Router1(config-if)#ip pim dense-mode
Router1(config-if)#end
Router1#
```

The second router looks remarkably similar:

```
Router2#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router2(config)#ip multicast-routing
Router2(config)#interface FastEthernet0/0
Router2(config-if)#ip address 192.168.3.1 255.255.255.0
Router2(config-if)#ip pim dense-mode
Router2(config-if)#exit
Router2(config)#interface Serial1/0
Router2(config-if)#ip address 192.168.2.6 255.255.255.252
Router2(config-if)#ip pim dense-mode
Router2(config-if)#end
Router2#
```

23.1.3 Discussion

With this simple configuration, you get all of the basic multicast functionality. The routers will distribute multicast packets properly, they will listen for end devices to join and leave groups with IGMP (Version 1, 2, or 3), and they will update one another with information about what multicast groups are currently in use, as well as where the servers and group members are. For many types of multicast applications, this is all you need. But it is important to remember that the PIM-DM protocol is appropriate only for certain types of networks with relatively specific multicast routing requirements.

First, PIM-DM works best in relatively small networks with no more than a few hops between the sender and the most remote receiver. Second, the number of multicast servers should be small, and the receivers should be scattered throughout the network in relatively large numbers. And third, because PIM-DM uses the multicast traffic itself to gather information about where the servers are, it needs a steady flow of traffic. In particular, it's a bad idea to use PIM-DM with multicast applications that can pause more than three minutes between packets—the routers will flush the routing information out of their tables and have to rebuild these tables when the next packet is received.

If one or more of these conditions is not true for any of your multicast applications, then you should probably consider one of the other routing protocols, particularly PIM-SM.

One final point to consider in any sort of routing is how the router will switch the packets. As we discussed in Chapter 11, you want to avoid process switching anything unless it's absolutely necessary. Fortunately, multicast packets are fast switched by default. However, in many configurations it is customary to disable multicast fast switching. So it is a good idea to look at your router configurations and make sure that you don't have any multicast interfaces that include the statement *no ip mroute-cache*. If any interfaces do have this command, then you should reenable the preferred default fast switching behavior using the interface level command, *ip mroute-cache*. This is true regardless of which multicast routing protocol you use.

Top

Recipe 23.2 Routing Multicast Traffic with PIMSM and BSR

23.2.1 Problem

You want to enable routing of multicasts using sparse mode for better efficiency, and use BSR for distributing RP information.

23.2.2 Solution

We've already discussed how PIM-SM requires a Rendezvous Point router. The most reliable way to achieve this is to have the network automatically discover the RP. This way, if the RP fails, another can automatically take over for it. We recommend using the Bootstrap Router (BSR) mechanism to dynamically distribute RP information.

There are two different types of router configurations for this type of network. Most of the routers will support end devices, both group members and servers. But a small number are configured to act as candidate RPs and candidate BSRs. In the example, we show the RP and BSR configuration in the same router. This isn't actually necessary, but it is convenient.

Router1 is an example of a "normal" multicast router. It forwards multicasts, takes part in PIM-SM, and may support group members or multicast servers as required:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#ip multicast-routing
Router1(config)#ip pim rp-address 192.168.15.5
Router1(config)#interface FastEthernet0/0
Router1(config-if)#ip address 192.168.1.1 255.255.255.0
Router1(config-if)#ip pim sparse-mode
Router1(config-if)#interface Serial1/0
Router1(config-if)#ip address 192.168.2.5 255.255.255.252
Router1(config-if)#ip pim sparse-mode
Router1(config-if)#end
Router1#
```

Router-RP1 is one of the candidate RPs, and it is also configured as a candidate BSR. It may also support multicast group members or servers if required. It is a good idea to configure two or more routers as RPs and BSRs in each multicast domain like this to provide redundancy:

```
Router-RP1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
```

```
Router-RP1(config)#ip multicast-routing
Router-RP1(config)#interface Loopback0
Router-RP1(config-if)#ip address 192.168.12.1 255.255.255.255
Router-RP1(config-if)# ip pim sparse-mode
Router-RP1(config-if)#exit
Router-RP1(config)#interface FastEthernet0/0
Router-RP1(config-if)#ip address 192.168.1.1 255.255.255.0
Router-RP1(config-if)#ip pim sparse-mode
Router-RP1(config-if)#exit
Router-RP1(config)#interface Serial1/0
Router-RP1(config-if)#ip address 192.168.2.5 255.255.255.252
Router-RP1(config-if)#ip pim sparse-mode
Router-RP1(config-if)#exit
Router-RP1(config)#ip pim rp-candidate loopback0 group-list 15
Router-RP1(config)#ip pim bsr-candidate loopback0 1
Router-RP1(config)#access-list 15 permit 239.5.5.0 0.0.0.255
Router-RP1(config)#access-list 15 deny any
Router-RP1(config)#end
Router-RP1#
```

23.2.3 Discussion

In larger networks, particularly networks with WAN links, the PIM-DM approach of forwarding all multicasts to all routers until they explicitly opt out of the group tends to be inefficient with network resources. So it is useful to configure a sparse mode multicast routing protocol such as PIM-SM. The basic configuration for most of the routers is similar to what we did in Recipe 23.1. The difference is that PIM-SM allows you to set up a RP router to act as the root of the multicast SPTs.

There are two ways to configure the routers to use an RP. The conceptually simpler method is to explicitly define the RP in the other routers, using the *ip pim rp-address* command, as shown in the previous configuration for Router1.

```
Router1(config)#ip pim rp-address 192.168.15.5
```

This method has two important administrative problems: if you ever want to change the RP to another router, you have to change it separately in every router, and it lacks the ability to automatically switch to a backup RP in case of failure. However, in the example, it is statically configured for a different reason, which we will explain in a moment.

The alternative is to configure the network to discover the RP dynamically, which is also shown in the solution. This is preferable in most cases. In fact there are two ways to accomplish this: one uses a Cisco proprietary method called Auto-RP, and the other uses the Bootstrap Router method, which is part of the open PIM-SM standard defined in RFC 2362. This recipe shows the Bootstrap Router method, which we generally prefer for interoperability reasons. The Auto-RP, which will only work in an all-Cisco network, is discussed in Recipe 23.3.

There are two important commands that define how Router RP1 will advertise itself. The first is *ip pim*

rp-candidate, which allows the router to advertise itself as a possible RP:

```
Router-RP1(config)#ip pim rp-candidate loopback0 group-list 15
```

There are two modifiers on this command. The first, loopback0, tells the network to use the loopback interface as the RP address. If there are several candidate RP routers, the PIM-SM algorithm prefers the one with the highest IP address. So be careful if you want a particular router to be the default.

This command also includes the *group-list* tag. In this case, it associates the RP functions for this router with access list 15, which specifies that this router will be the RP for any multicast group between `239.5.5.0` and `239.5.5.255`. You may specify that the RP router can support any set of multicast group addresses. If you want it to support all multicast groups, simply eliminate the *group-list* keyword:

```
Router-RP1(config)#ip pim rp-candidate loopback0
```

Using the group list option is most useful if you have an extremely large number of multicast groups to distribute and the load is too heavy for one router. Some network administrators may also decide to use a different RP for a few specific local groups for ease of management as well. But in most networks, there is relatively little benefit to breaking up the RP functions by group this way. We have just included the option here to show how it works in case you do need it. If you decide to break up the RP responsibilities among many routers, be careful to ensure that all possible groups have an RP available.

The next important command in Router RP1 is*ip pim bsr-candidate*:

```
Router-RP1(config)#ip pim bsr-candidate loopback0 1
```

This allows the router to act as a Bootstrap Router (BSR). BSRs are responsible for distributing information about all of the known candidate RPs throughout the network. In this example, we use the loopback interface to define the IP address that this router will use when advertising itself as a BSR candidate. The protocol uses this address in the election process to select the BSR from all of the possible candidates. This command also accepts a *priority* keyword that you can use to help bias the BSR election process.

The last number, 1, in the example's*bsr-candidate* command is a hash value that helps to select different RPs for different ranges of multicast group addresses. Because the example uses a range of addresses from `239.5.5.0` to `239.5.5.255`, we could have configured the candidate BSR to allow RPs to control a similar range of multicast addresses. There are 8 bits of freedom in this range, which would give a hash value of 8. If you're not sure what to use here, it is safest to just use a value of 1 bit. This will allow the network to select the best RPs on a group-by-group basis. In fact, this option is really useful only when you have several RPs, each supporting different ranges of multicast group addresses. When in doubt, it is safe to use a value of 1. There is a slight performance advantage to using larger hash values, however.

We have one final comment on the configuration of the candidate RP/BSR router. Since we are using the loopback interface as the source for both RP and BSR functions, it is important that this interface be configured for PIM-SM. This seems counter-intuitive because a loopback interface can't have any

neighbors by definition. But the BSR function in particular will not work properly without this configuration, because we use the loopback interface to define this router as a BSR candidate.

We use the loopback interface for this purpose because it can't go down. If there is any path to this router, it will retain the RP role. This may not always be desirable, of course. If this is a WAN router, for example, we certainly wouldn't want all multicast traffic to have to cross the WAN twice just because an Ethernet interface went down. In such cases, it would be better to use the Ethernet port for the BSR and RP addresses.

In general, it is a good practice to set up several BSRs to spread the word about several RPs with overlapping group ranges for redundancy. This gives a much more reliable network. And, if it ever happens that no RPs are available, the router will revert to the statically configured RP address, which we have configured using the following command:

```
Router1(config)#ip pim rp-address 192.168.15.5
```

Note that the address specified is not the same as Router RP1. The static RP value is needed only when none of the usual RPs are available. So a good choice for this last resort RP would be one of your central core routers. Naturally, you must make sure that the router you specify is configured to act as an RP.

If you are going to use this BSR method for RP discovery, you need to take certain network design precautions. Sometimes you will want an RP to serve a particular section of the network. In smaller networks, you may have RPs serve the entire physical expanse of the network, but with different RPs serving different multicast groups. However, in larger networks, or when two or more networks adjoin, it is necessary to limit the region of the network served by any RP.

The BSR works by sending out information to all of the adjacent PIM routers. These routers record all of the information, then relay the same information to all of the adjacent routers (except for the one that originally provided the information). Because this process is repeated at each hop, it could expand indefinitely. In fact, the process is not even bounded by the usual IP TTL limit of 255 hops because a new packet is created at each hop. So it is possible to have the network choose an RP that some devices cannot reach, particularly if you use TTL to control multicast scope, as in Recipe 23.10.

To define distinct regions of a network served by different groups of RPs, you first need to decide where the boundaries of these regions will be, then configure the routers along the boundary so that they will neither transmit nor receive any BSR information on those interfaces:

```
Router-Border1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router-Border1(config)#ip multicast-routing
Router-Border1(config-if)#interface FastEthernet0/0
Router-Border1(config-if)#ip pim sparse-mode
Router-Border1(config-if)#ip pim border
Router-Border1(config-if)#end
Router-Border1#
```

Note that the *ip pim border* command affects only the exchange of BSR information. Multicast traffic can still flow across the interface, and PIM will still form SPT trees that cross the interface.

Top

Recipe 23.3 Routing Multicast Traffic with PIM-SM and Auto-RP

23.3.1 Problem

You want to allow routing of multicasts using sparse mode, and use Auto-RP for distributing RP information.

23.3.2 Solution

This recipe accomplishes the same basic tasks as Recipe 23.2, but using a different method. If you are unfamiliar with PIM-SM, please read that recipe first. There are two different types of router configurations for Auto-RP configuration, just as there are for BSR. Router1 represents a regular multicast-enabled router anywhere in the network. This router supports end devices as group members or servers, as well as routing multicast traffic for other routers:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#ip multicast-routing
Router1(config)#ip pim rp-address 192.168.15.5
Router1(config)#interface FastEthernet0/0
Router1(config-if)#ip address 192.168.1.1 255.255.255.0
Router1(config-if)#ip pim sparse-dense-mode
Router1(config-if)#exit
Router1(config)#interface Serial1/0
Router1(config-if)#ip address 192.168.2.5 255.255.255.252
Router1(config-if)#ip pim sparse-dense-mode
Router1(config-if)#end
Router1#
```

The second type of configuration is for a candidate RP router, called Router-RP1. This router may also support group members or servers. As in the previous recipe, it is a good idea to configure two or more routers in each multicast domain like this to provide redundancy:

```
Router-RP1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router-RP1(config)#ip multicast-routing
Router-RP1(config)#interface Loopback0
Router-RP1(config-if)#ip address 192.168.12.1 255.255.255.255
Router-RP1(config-if)#ip pim sparse-dense-mode
Router-RP1(config-if)#exit
Router-RP1(config)#interface FastEthernet0/0
Router-RP1(config-if)#ip address 192.168.1.1 255.255.255.0
```

```
Router-RP1(config-if)#ip pim sparse-dense-mode
Router-RP1(config-if)#exit
Router-RP1(config)#interface Serial1/0
Router-RP1(config-if)#ip address 192.168.2.5 255.255.255.252
Router-RP1(config-if)#ip pim sparse-dense-mode
Router-RP1(config-if)#exit
Router-RP1(config)#ip pim send-rp-announce loopback0 scope 16 group-list 15
Router-RP1(config)#ip pim send-rp-discovery scope 16
Router-RP1(config)#access-list 15 permit 239.5.5.0 0.0.0.255
Router-RP1(config)#access-list 15 deny any
Router-RP1(config)#end
Router-RP1#
```

23.3.3 Discussion

Recipe 23.2 discussed one way of discovering the RPs in a PIM-SM network using the BSR method. This recipe shows an alternative method. The BSR method requires Version 2 of the PIM-SM protocol, which Cisco started supporting in IOS 11.3T. If you have earlier IOS versions in your multicast network, you will need Auto-RP, perhaps used in parallel with BSR. As long as both systems select the same RPs, you should be able to simultaneously run both methods. However, if you do run into interoperability problems, you can disable the Version 2 functionality on newer routers using the following command:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#ip pim version 1
Router1(config)#end
Router1#
```

Auto-RP distributes information about the multicast RPs using the globally registered multicast group addresses `224.0.1.39` and `224.0.1.40`. This is an elegant solution to the problem of how to tell the network where the RPs are. But it presents a bit of a paradox to the routers: how can they distribute multicast information using PIM-SM if they don't yet have an RP? Cisco solved this problem by creating a new hybrid PIM mode called sparse-dense mode. This means that the routers should use sparse mode if there is a known RP, and dense mode if there isn't. So the only difference in Router1's configuration between this recipe and the previous one is that all of the interfaces are configured with the command *ip pim sparse-dense-mode*:

```
Router1(config-if)#ip pim sparse-dense-mode
```

There are two important differences between Router RP1's configuration in this recipe and in Recipe 23.2. To advertise its willingness to become an RP using Auto-RP, this router includes the global configuration command *ip pim send-rp-announce*:

```
Router-RP1(config)#ip pim send-rp-announce loopback0 scope 16 group-list 15
```

The interface specified in this command, `loopback0`, has the address that other routers will use for all of their interactions with the RP. We have used a loopback interface to ensure that, as long as there is

an active path to this router, it can continue to act as the RP.

As mentioned in Recipe 23.2, however, this may not always be desirable. For example, if this router has a LAN interface and a WAN interface, you certainly don't want all of your multicast traffic to have to loop through the WAN if the LAN interface goes down. In such failure modes, you would probably want to stop using this RP and switch to a different candidate RP. You can do this by simply specifying the LAN interface in the *send-rp-announce* command.

The second difference in Router RP1's configuration is the command *ip pim send-rp-discovery*:

```
Router-RP1(config)#ip pim send-rp-discovery scope 16
```

This instructs the router to act not only as a candidate RP, but also as an RP mapping agent. The mapping agent function is similar to the BSR function that we discussed in Recipe 23.2. This is the router that is responsible for distributing information about all of the RPs throughout the network. Although you could make the mapping agent a different router, we have combined the functions on the candidate RP router for the same reasons as in the BSR case.

Note that for both the *send-rp-announce* and *send-rp-discovery* commands, there is a *scope* keyword that sets the TTL scope for these functions to 16. Because Auto-RP uses multicasts to distribute its information, you can specify a particular initial TTL value. This controls the network area that will be able to use this particular RP. Recipe 23.10 includes a detailed discussion of how to use TTL for controlling multicast scope.

Finally, the *send-rp-announce* command specifies a *group-list* keyword. This is identical to the group list in the BSR configuration of Recipe 23.2. It defines the groups for which this particular router is willing to act as RP. As we mentioned in Recipe 23.2, most networks can easily support all of their multicast traffic on a single active RP. Having different RPs for different multicast groups is primarily useful for administrative reasons, and for rare networks that have too many multicast groups for one RP to support.

If you want one RP for all groups, simply leave out the *group-list* keyword and its arguments:

```
Router-RP1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router-RP1(config)#ip pim send-rp-announce loopback0 scope 16
Router-RP1(config)#ip pim send-rp-discovery scope 16
Router-RP1(config)#end
Router-RP1#
```

23.3.4 See Also

Recipe 23.2; Recipe 23.10

Top

Recipe 23.4 Configuring Routing for a Low Frequency Multicast Application

23.4.1 Problem

You have a multicast application where the servers send packets less frequently than the standard PIM timeout intervals.

23.4.2 Solution

PIM-SM is best suited to this type of application. The configurations of the RP and BSR or Auto-RP routers for this example are identical to those shown in Recipe 23.2 and Recipe 23.3. The differences appear on the other routers in the network. So this recipe shows only the configurations for these other routers:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#ip multicast-routing
Router1(config)#ip pim spt-threshold 10 group-list 15
Router1(config)#access-list 15 permit 239.5.5.55
Router1(config)#access-list 15 deny any
Router1(config)#interface FastEthernet0/0
Router1(config-if)#ip address 192.168.1.1 255.255.255.0
Router1(config-if)#ip pim sparse-dense-mode
Router1(config-if)#exit
Router1(config)#interface Serial1/0
Router1(config-if)#ip address 192.168.2.5 255.255.255.252
Router1(config-if)#ip pim sparse-mode
Router1(config-if)#end
Router1#
```

23.4.3 Discussion

Discussions of multicast applications usually focus on high-bandwidth applications. But some multicast applications have the opposite behavior. For example, some news broadcast type applications allow message servers to send short messages to a large group of users. This sort of service might be used for administrative purposes ("the server is going down in five minutes") or for business purposes ("stop selling widgets, the warehouse is empty," or "buy more Cisco stock"). In either case, all of the users in the group need to receive the message.

But there are three problems. First, in this sort of application you can't afford to waste the first packet

while setting up your multicast distribution trees, because it may be the only packet sent. Second, you don't want to worry about finding a more optimal tree after you've delivered the first packet, because there isn't going to be another packet for a long time. And third, by the time the next packet comes along, all of the routing that the network set up for the first packet may have timed out.

There are many possible solutions to this problem. You could statically configure the entire multicast network using static multicast routes and group memberships, and not use a multicast routing protocol at all. But updating this is clearly an administrative nightmare, and it would make it difficult to use the same network for other multicast applications. This recipe shows a better solution.

It is useful to step through the network from group member to sender and back to see all of the places where a low-frequency multicast application can cause problems. A router will assume that there are no group members on a network segment if it doesn't receive any IGMP messages for a defined period of time. However, the devices on the segment are responsible for sending periodic IGMP reports to ensure that this doesn't happen unless there really aren't any more members for a given group on the segment. So, unless the timers are not configured properly on the end devices, the low-frequency multicast application should not represent a problem here. In particular, it should not be necessary to configure a static IGMP statement on the router interface. If it turns out that IGMP timeouts are a problem for an application, refer to Recipe 23.6 for information about setting up static IGMP statements on a router.

At this point, it is useful to think about whether you should use a dense mode or sparse mode multicast routing protocol. Recall that the difference between these modes is that a dense mode protocol forwards all multicast packets to all neighboring routers until they announce that they don't want to be members. In sparse mode, on the other hand, routers must explicitly join a group before they can receive it.

If you were to use PIM-DM in this network, the multicast packets themselves would help to establish the multicast tree structure. If a router didn't receive any packets for a group within a standard timeout period of three minutes, it would tear down this tree structure. So this is the first important place where the low-frequency application can cause problems. The network either has to be constructed so that it can build and tear down this forwarding tree for every individual multicast packet, or it has to keep the tree up.

All of the end devices in the low-frequency application prejoin the group, so before the first packet is sent, it is already clear where it should be delivered. If you used dense mode, the routers that did not have members would respond to the first packet with a round of Prune messages asking to be removed from the tree. Since we don't expect another packet any time soon, these Prune messages are really just a waste of network resources. The entire tree structure will have been removed before the next packet arrives. So, all things considered, it is better to use PIM-SM than PIM-DM for this type of application. Note that the example actually uses sparse-dense mode so that it can function in dense mode if an RP is not available. This is discussed in Recipe 23.3.

As discussed in the introduction to this chapter, every PIM-SM router sends a fresh Join up the tree towards the RP once per minute. This ensures that the tree to the RP stays up even if there is no traffic.

When the source device is finally ready to send a packet, it sends this packet to its first-hop router. The first-hop router encapsulates the multicast packet in a unicast registration packet that it sends to the RP. The RP pulls the multicast packet back out of the encapsulation and forwards it down the multicast tree to all of the group members. It then extends the tree for this group over to the source so that it can use multicast all the way on subsequent packets.

The only impact of the low-frequency nature of the application here is that the extension of the tree from the RP over to the source will eventually be torn down because of lack of data. But we do need to be careful that the last-hop routers don't try to build a new multicast-forwarding tree rooted at the source. To prevent this, the recipe example shows how to set the SPT threshold value so that only high traffic multicast streams will rebuild the tree like this.

This threshold is set by the command *ip pim spt-threshold*:

```
Router1(config)#ip pim spt-threshold 10 group-list 15
```

In the example, we set the threshold to 10Kbps for the groups defined by access list 15. All other groups will continue to use the default threshold of zero. If you instead wanted to have this command affect all multicast groups, you could simply leave out the *group-list* command:

```
Router1(config)#ip pim spt-threshold 10
```

You can also use this command to tell the routers to always use the RP, and never switch over to the source-based SPT by using the keyword *infinity* for the threshold value:

```
Router1(config)#ip pim spt-threshold infinity
```

Or, you could use this option with a *group-list* statement to prevent the router from building a source-based SPT just for a specific set of groups:

```
Router1(config)#ip pim spt-threshold infinity group-list 15
```

23.4.4 See Also

Recipe 23.2; Recipe 23.3; Recipe 23.6

Top

Recipe 23.5 Configuring CGMP

23.5.1 Problem

You want the router to use CGMP to communicate with a Catalyst switch.

23.5.2 Solution

When you enable multicast routing and turn on PIM on an interface, IGMP is enabled by default. However, you must explicitly enable CGMP on the router if you want your Catalyst switch to take advantage of this efficient way of handling group membership:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#ip multicast-routing
Router1(config)#interface FastEthernet0/0
Router1(config-if)#ip pim sparse-dense-mode
Router1(config-if)#ip cgmp
Router1(config-if)#end
Router1#
```

23.5.3 Discussion

The introduction to this chapter discusses the reasons for CGMP and roughly how it works. It is important to remember that CGMP is a Cisco proprietary protocol, so it will only work with Catalyst switches. Table 23-1 shows the minimum switch software revision level required for CGMP for several types of switches. If in doubt, please check your switch's documentation.

Table 23-1. Catalyst switches that support CGMP

Catalyst model	Minimum software version
1900	6.0
2820	6.0
2900XL	11.2(8)SA

Catalyst model	Minimum software version
2901, 2902, 2926T/F/G	2.3
2948G	All
3500XL	11.2(8)SA
4000 series	All
4912	All
5000	2.3
6000	Not supported

Top

Recipe 23.6 Static Multicast Routes and Group Memberships

23.6.1 Problem

You want to override the dynamic multicast routing and group membership with static entries.

23.6.2 Solution

By default, PIM will use the same dynamic routing table learned by the unicast routing protocol. However, in some cases you don't want to use these routes. For example, you might have to send multicast traffic through a tunnel to bypass a section of network that doesn't support multicast routing. In this case, the unicast routing table is clearly the wrong path for multicast traffic. So you need to specify a different route for multicast traffic to use:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#ip multicast-routing
Router1(config)#ip mroute 192.168.15.0 255.255.255.0 192.168.98.6
Router1(config)#interface Tunnel0
Router1(config-if)#ip address 192.168.98.5 255.255.255.252
Router1(config-if)#ip pim sparse-dense-mode
Router1(config-if)#tunnel mode gre ip
Router1(config-if)#end
Router1#
```

You can specify a static IGMP Join to ensure that the router always thinks that there are group members on an interface:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#ip multicast-routing
Router1(config)#interface FastEthernet0/0
Router1(config-if)#ip pim sparse-dense-mode
Router1(config-if)#ip igmp join-group 239.5.5.55
Router1(config-if)#end
Router1#
```

23.6.3 Discussion

The static *mroute* command is used only to describe the Reverse Path Forwarding (RPF) the multicast traffic should take. PIM doesn't redistribute this information, but all devices on the internal network

need to know that this router is the gateway to this particular external network so that they can also build their SPT trees appropriately. It is likely that other routers will also need static *mroutes* if you are using tunnels like this.

Static multicast routes are most frequently used when the unicast network doesn't have the same topology as the multicast network. There are two main reasons why this might be the case. The recipe example suggests one of these reasons: there may be tunnels that bypass non-multicast sections of the network.

The other common reason for using a static multicast route is to force multicast traffic to take a different path than unicast traffic. For example you might have a separate network link for multicast traffic. As with all static routing, doing this creates administrative problems because it is very difficult to construct static routes that adapt to network failures.

The static IGMP Join example is most useful when there are devices on the segment that may have poor IGMP implementations or when they Join and Leave extremely rapidly. Alternatively, as in Recipe 23.14, the receiving devices may not know that this is a multicast application. A static IGMP statement ensures that the router always thinks that there are group members on this interface.

You should be careful about using this command on any links that contain other routers because it may lead to multicast routing loops. This is because the router will always forward packets for this group out this interface, even if PIM would normally prune this link from the tree.

23.6.4 See Also

Recipe 23.14

Top

Recipe 23.7 Routing Multicast Traffic with MOSPF

23.7.1 Problem

You want to distribute your multicast routing tables with MOSPF.

23.7.2 Solution

Unfortunately, Cisco does not support MOSPF. As mentioned in the introduction to this chapter, MOSPF is a set of multicast extensions to OSPF that uses LSA Type 6. By default, when a Cisco router receives a Type 6 LSA packet it will generate a "%OSPF-4-BADLSATYPE" error message. To avoid this error message, you can configure your routers to ignore Type 6 LSA packets:

```
Router#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router(config)#router ospf 65530
Router(config-router)#ospf ignore lsa mospf
Router(config-router)#end
Router#
```

23.7.3 Discussion

MOSPF has not enjoyed a particularly wide acceptance for several reasons, mostly related to the fact that it uses a dense-mode multicast forwarding scheme, and because it is protocol dependent. It turns out to be most useful in networks that meet several key requirements:

- They use a relatively small number of multicast applications.
- These applications have few servers and many group members, with the group members scattered throughout the network.
- The network must use OSPF as its unicast protocol.
- The applications deliver a flow of multicast traffic that is neither heavy enough to cause congestion problems on the slowest links in the network, nor so light that relationships time out in the routers.

Few router vendors have implemented MOSPF. Increasingly, the multicast routing protocol of choice appears to be PIM-SM.

However, it is important to note that the normal default behavior for PIM is to use the standard IP unicast routing table. If you happen to be using OSPF for your standard IP routing tables, PIM will use the same OSPF routing tables to construct its RPF trees back to the multicast source or RP by default. So you can use OSPF with PIM, without any additional configuration.

The only thing that you miss by not being able to use MOSPF is the ability to dynamically distribute a multicast routing table that reflects a different topology than the unicast routing table.

23.7.4 See Also

Chapter 8

Top

Recipe 23.8 Routing Multicast Traffic with DVMRP

23.8.1 Problem

You want to route multicast traffic using the DVMRP protocol.

23.8.2 Solution

Cisco routers support DVMRP only as a gateway to PIM. So the configuration is remarkably similar to the PIM configuration. The key difference is in the *ip dvmrp unicast-routing* command, which tells the router to use the DVMRP multicast routing table instead of the usual PIM choice of the unicast routing table:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#ip multicast-routing
Router1(config)#interface FastEthernet0/0
Router1(config-if)#ip pim sparse-dense-mode
Router1(config-if)#ip dvmrp unicast-routing
Router1(config-if)#ip dvmrp summary-address 192.168.0.0 255.255.0.0
Router1(config-if)#end
Router1#
```

23.8.3 Discussion

Before saying anything else, we must stress once again that DVMRP is not our first choice for a multicast routing protocol. PIM-DM is simpler to implement and more efficient with router resources if you want a dense-mode protocol. And PIM-SM scales much better in larger networks or when there are many groups. In fact, Cisco routers do not provide a full implementation of DVMRP. They can take part in DVMRP neighbor discovery and route exchange, but internally the multicast processing is done using PIM. However, you can configure PIM to use the DVMRP routing table, which is what this recipe actually does.

DVMRP is defined in RFC 1075, and was the first widely implemented multicast routing protocol, forming the core of the Internet's original Multicast Backbone (MBONE). So the main reasons for implementing it today are to give compatibility with an existing DVMRP network. It is unlikely that anybody would want to implement a new production DVMRP network today, just as few people would want to implement any other routing protocol that hasn't been significantly updated since 1988 (the original publication date of RFC 1075).

The DVMRP protocol differs from PIM-DM in a fundamental way. While PIM-DM uses the router's existing unicast routing table to determine the shortest path back to the source, DVMRP distributes and maintains its own internal routing table. This was needed because early multicast networks tended to use a lot of tunnels to allow multicast traffic to cross large network regions that supported only unicast routing. So the multicast network topology could be quite different from the unicast topology, necessitating a separate routing table.

DVMRP distributes routing information between adjacent routers by means of multicast packets sent on the globally defined, well-known multicast address `224.0.0.4`. This process begins by exchanging neighbor information. Every router needs to know about all of its neighbors so that it can always determine the best path back to the source, which is the only way to eliminate loops. This neighbor information is updated every 60 seconds by default. If a neighbor misses four consecutive updates, the router considers it dead and removes it from the routing tables.

These periodic updates also contain all of the multicast routing information for the network. If a route is not seen in two consecutive updates, then it is considered invalid and is no longer used. It is removed from the tables if it remains invalid for two additional minutes. So, when there are network problems, it is possible to have interruptions of up to two minutes.

Cisco does not offer a full DVMRP implementation. Instead, it gives the ability to make a gateway between DVMRP and PIM. This allows a Cisco router to perform many of the standard DVMRP functions, but it makes it impossible to use Cisco routers to build a pure DVMRP network. In particular, Cisco routers do not send any DVMRP Probe messages to establish new neighbor relationships.

However, when you enable PIM routing on an interface, the router also starts to listen for DVMRP announcements addressed to group `224.0.0.4`. The Cisco router responds to these announcements by exchanging its routing tables with the DVMRP neighbors. However, by default, it does not use the DVMRP routing table it receives unless the interface has the command *ip dvmrp unicast-routing* enabled. This is particularly important in cases where the multicast topology is different than the unicast network topology (the DVMRP tunnel example in Recipe 23.9 is one such case).

Since DVMRP functionality on Cisco routers generally represents a gateway between pockets of PIM and DVMRP functionality, we have included another useful command in the recipe example. The command *ip dvmrp summary-address* tells the router to advertise only a summary route into the DVMRP network:

```
Router1(config-if)#ip dvmrp summary-address  192.168.0.0 255.255.0.0
```

This helps to simplify the multicast routing tables to save memory and improve overall efficiency.

In some cases, you might have a router connected to a segment with several DVMRP devices, such as Unix servers running *mrouted*. There are two ways that you can restrict what your router accepts from DVMRP devices. Both are variants of the interface configuration command *ip dvmrp accept-filter*:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#ip multicast-routing
Router1(config)#access-list 11 permit 192.168.1.17 0.0.0.0
Router1(config)#access-list 11 permit 192.168.1.18 0.0.0.0
Router1(config)#access-list 11 deny any
Router1(config)#interface FastEthernet0/0
Router1(config-if)#ip pim sparse-dense-mode
Router1(config-if)#ip dvmrp unicast-routing
Router1(config-if)#ip dvmrp accept-filter 0 neighbor-list 11
Router1(config-if)#end
Router1#
```

This version of the *ip dvmrp accept-filter* command restricts the allowed DVMRP neighbors. If this router receives DVMRP updates from any devices not in the specified access list, it will simply ignore them.

The other version of this command specifies what ranges of multicast source address ranges this router will accept:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#ip multicast-routing
Router1(config)#access-list 12 permit 192.168.10.0 0.0.0.255
Router1(config)#access-list 12 deny any
Router1(config)#interface FastEthernet0/0
Router1(config-if)#ip pim sparse-dense-mode
Router1(config-if)#ip dvmrp unicast-routing
Router1(config-if)#ip dvmrp accept-filter 12 95
Router1(config-if)#end
Router1#
```

In this case, the router is willing to listen only to information about sources on the `192.168.10.0/24` network. And we have gone slightly further by specifying an administrative distance of 95 for these routes. When it hears about these multicast routes, this router will automatically give them a distance of 95. This will ensure that if these same routes are learned by another method that has a better administrative distance, the router will not use them. For example, in Chapter 5 we saw that EIGRP has a default administrative distance value of 90, while OSPF has a value of 110. So this command will tell the router to use the DVMRP routes for constructing source-rooted multicast trees if the unicast route it has for the source is from OSPF. But, if it has an EIGRP unicast route, it will use that route instead.

Note that the router will never use the DVMRP route for forwarding unicast traffic. It only uses this information for constructing loop-free trees back to the multicast source, which in turn tells the router which interfaces it can forward multicast traffic through.

23.8.4 See Also

Recipe 23.1, Recipe 23.9; Chapter 5; Chapter 6; RFC 1075

Top

Recipe 23.9 DVMRP Tunnels

23.9.1 Problem

You want to create a DVMRP tunnel to bypass a section of network that doesn't support multicast routing.

23.9.2 Solution

You can create a DVMRP tunnel from a Cisco router to a non-Cisco DVMRP device using the special DVMRP tunnel mode. This allows you to pass multicast traffic through a section of network that doesn't support multicast routing:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#ip multicast-routing
Router1(config)#interface Tunnel0
Router1(config-if)#ip unnumbered FastEthernet0/0
Router1(config-if)#ip pim sparse-dense-mode
Router1(config-if)#ip dvmrp unicast-routing
Router1(config-if)#tunnel source FastEthernet0/0
Router1(config-if)#tunnel destination 192.168.99.15
Router1(config-if)#tunnel mode dvmrp
Router1(config-if)#exit
Router1(config)#interface FastEthernet0/0
Router1(config-if)#ip address 192.168.1.1 255.255.255.0
Router1(config-if)#ip pim sparse-dense-mode
Router1(config-if)#end
Router1#
```

23.9.3 Discussion

There are several important things to remember about DVMRP tunnels. First, you cannot create a DVMRP tunnel between two Cisco routers because neither router will send DVMRP Probe messages. This feature is used to create tunnels only between Cisco routers and native DVMRP devices, such as non-Cisco routers or Unix servers running *mrouted*.

Second, a DVMRP tunnel is essentially just a standard IP-in-IP tunnel. However, it is only used to transmit multicast traffic. Unicast traffic follows the normal network path. This is markedly different from several of the examples of GRE tunneling (a more flexible variant of IP-in-IP tunneling)

discussed in Chapter 12, where the tunnel was used to carry unicast traffic.

Third, the device on the other end of this tunnel must also be explicitly configured for this tunnel. The IP address it specifies for the tunnel destination must match the tunnel source address specified on the Cisco router. In this case, the source is the interface `FastEthernet0/0`, so the device on the other end must specify a tunnel destination address of `192.168.1.1`, which is this interface's IP address. Similarly, the address specified as the tunnel source on the other device must match the tunnel destination address on the Cisco end.

Fourth, both the tunnel interface and source interface must have PIM enabled. We have used sparse-dense mode for PIM, which is just to cover all of the possibilities. But the router will not route multicast traffic correctly without some form of PIM enabled on these interfaces.

Finally, if you want to create a PIM tunnel to another Cisco device, the easiest way to do so is to create a GRE tunnel and use a static *mroute*, as we did in Recipe 23.6. This will ensure that multicast traffic uses the tunnel, rather than trying to use the unicast path. A DVMRP tunnel can only connect a Cisco router to a non-Cisco native DVMRP device.

23.9.4 See Also

Recipe 23.6; Recipe 23.8; Chapter 12

Top

Recipe 23.10 Controlling Multicast Scope with TTL

23.10.1 Problem

You want to ensure that your multicast traffic remains confined to a small part of the network.

23.10.2 Solution

You can define a TTL threshold value for each interface on a router. The *ttl-threshold* command instructs the router to drop any multicast packets that have a TTL value less than or equal to the specified value. The router will receive packets on this interface normally. This command affects only the transmission of multicast packets:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#ip multicast-routing
Router1(config)#interface FastEthernet0/0
Router1(config-if)#ip multicast ttl-threshold 16
Router1(config-if)#end
Router1#
```

This technique works for all multicast routing protocols.

23.10.3 Discussion

The first popular method for controlling the scope of multicast transmissions made use of the standard IP TTL header variable. This is an 8-bit variable that each router decrements by one as it forwards the packet. If a router receives a packet with a TTL value of 1, it won't forward it any further. The main use of TTL in unicast IP networking is to help to kill loops. In a loop situation a packet may go around a few times, but the TTL value keeps decrementing and will eventually reach 1. The network will then drop it.

In multicast networking, TTL has a more subtle use. Most multicast applications include the ability to define an initial TTL value other than the default 255. This leads to a common method for keeping multicast traffic within a certain well-defined part of the network. The customary technique is to configure the multicast applications with an initial TTL value that defines the scope, as shown in Table 23-2.

Table 23-2. Commonly used TTL values for controlling multicast scope

Scope	Initial TTL value
Local segment	1
Site, department or division	16
Enterprise	64
World	128

Configure these values on your multicast server to define how far you want the application to reach. Then enforce these limits with your routers. For multicast traffic that is purely local to the server's network segment, there is no need to do anything special on the routers. When a router receives a packet with a TTL value of 1, it decrements this value to get 0 and drops the packet without forwarding it any further.

The previous example shows how you would configure the routers along the boundary between two departments. If a department has a multicast application that is intended to serve only this department, they would configure the routers that connect to other departments or to the network backbone to drop any multicast packets with a TTL value less than or equal to 16, as shown in the example:

```
Router1(config-if)#ip multicast ttl-threshold 16
```

It doesn't matter where in the internal departmental network this server is located; the boundary routers will always prevent these packets from reaching the rest of the network. This way, another department in the same enterprise can have an application using the same multicast group address without conflict.

Similarly, if a multicast application serves an entire enterprise network, you would configure the server to use an initial TTL value of 64. Then you would configure the border around the edges of your enterprise network to drop any multicast packets with a TTL value greater than or equal to 64.

The reason for suggesting an initial TTL value of 128 for worldwide applications is simply to allow for future implementation of new multicast domains.

The TTL values shown in Table 23-2 are just suggestions based on common usage. There is no mandated standard, nor is there an IETF best current practices document on this subject.

There are a couple of important problems with using TTL thresholds to control multicast scope. The most critical is that you must rely on the server configuration. If a new server is turned on to offer multicast services within a network, you have to cross your fingers and hope that the system's administrators have configured the initial TTL values properly to prevent leakage. The worst case is when two or more departments use applications with the same multicast group numbers. If the traffic from one manages to leak to the other, it can cause serious confusion to the application. It is also relatively easy to have conflicts between global and local multicast applications using the same group

addresses.

TTL scoping can also cause serious problems for multicast routing protocols at the network boundaries. The routers at these boundaries will have trouble pruning themselves from unwanted SPT trees in dense-mode routing because they effectively act as a sink for all multicast traffic. This can cause CPU problems on the border routers as they handle extra multicast traffic only to drop it.

For these reasons, it is generally better to use Administratively Scoped Addressing for controlling multicast scope, as shown in Recipe 23.11.

Note that when a router needs to drop a unicast packet because of a TTL limitation, it sends an ICMP TTL exceeded error message back to the source. However, according to RFC 1812, such messages are not generated for multicast packets. This is good because it would otherwise cause a continuous barrage of ICMP errors from the perimeter of the network that could potentially cause network congestion and CPU problems on the multicast source device.

23.10.4 See Also

Recipe 23.11; RFC 1812

Top

Recipe 23.11 Using Administratively Scoped Addressing

23.11.1 Problem

You want to use RFC 2365 administratively scoped multicast addressing to control how multicast traffic is distributed through your network.

23.11.2 Solution

To configure regions of multicast scope using addressing rather than TTL, use the *ip multicast boundary* interface command:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#ip multicast-routing
Router1(config)#access-list 15 deny 239.255.0.0 0.0.255.255
Router1(config)#access-list 15 permit any
Router1(config)#interface FastEthernet0/0
Router1(config-if)#ip multicast boundary 15
Router1(config-if)#end
Router1#
```

Note that the access list uses a *deny* statement to specify which groups are to be dropped. Because access lists have an implicit deny on all addresses not explicitly matched, you must include a *permit ip any* at the end of the access list to allow all other multicast groups to pass normally.

23.11.3 Discussion

RFC 2365 defined a new way of handling the problem of controlling the scope of multicast applications. The IETF realized that you can't count on applications to have the right initial TTL value. The other important reason that they give for avoiding TTL scoping has to do with pruning in dense mode multicast protocols such as PIM-DM or DVMRP.

The problem is that the router at the boundary is dropping all multicast packets, but it can't tell its upstream neighbors that it no longer wants to be a member of this SPT. And it isn't sufficient to simply discover the problem and prune the tree at the previous node because of the possibility that the packets will still reach the destination by another path. The result is that the routers at the TTL boundary become *sinks* that receive all multicast traffic but cannot process it. This can cause performance

problems if there is a lot of multicast traffic.

Another problem is caused by misconfigured multicast servers causing multicast applications to leak out of their domains. This problem can occur either accidentally or deliberately, and in either case the result is that multicast applications in neighboring network regions will not work properly. A similar problem occurs when the routers at the boundary are not configured properly, allowing even well-behaved multicast applications to leak out of their domains. Clearly a better solution is required.

The alternative proposed in RFC 2365 is to set aside particular ranges of multicast group addresses for unregistered use, similar to the RFC 1918 system of unregistered IP addresses. The RFC takes the range from `239.0.0.0` to `239.255.255.255` for this purpose. It defines how to use particular ranges of these addresses for multicast applications with different scopes. These ranges are summarized in Table 23-3.

Table 23-3. Administratively scoped multicast addressing

Scope	Address range	Access list
Site Local	239.255.0.0/16	access list 15 deny 239.255.0.0 0.0.255.255
Expanding Site Local	239.254.0.0/16, 239.253.0.0/16, etc.	access list 15 deny 239.254.0.0 0.0.255.255 access list 15 deny 239.253.0.0 0.0.255.255, etc.
Organization Local	239.192.0.0/14	access list 15 deny 239.192.0.0 0.3.255.255
Expanding Organization Local	239.128.0.0/10, 239.64.0.0/10, 239.0.0.0/10	access list 15 deny 239.128.0.0 0.63.255.255 access list 15 deny 239.64.0.0 0.63.255.255 access list 15 deny 239.0.0.0 0.63.255.255

The IANA has registered several other multicast addresses for specific applications. The range from `224.0.0.0` through `224.0.0.255` is reserved for routing protocols and other network maintenance applications that are confined to the local network segment. The multicast addresses between `224.0.1.0` and `224.0.1.255` have been designated for Internetwork control applications such as Cisco's RP discovery protocol. And the range from `224.0.2.0` through `224.0.255.255` is set aside for miscellaneous well-known registered application purposes. Please refer to the IANA web site (http://www.iana.org/assignments/multicast-addresses) for specific well-known addresses.

The configuration example shows a Site Local boundary. The *ip multicast boundary* command affects the packets sent and received on this interface. So, you would configure this command on all interfaces that connect to other sites. For interfaces that connect to other organizations the only difference is in the access list, as shown in Table 23-3.

If you needed more multicast address space, you could add `239.254.0.0` first, then `239.253.0.0`, and so forth.

In many organizations you will see TTL (see Recipe 23.10) and administratively scoped addressing used simultaneously to control the reach of multicast applications. The two methods work well together, which makes it relatively easy to accomplish a transition from one method to the other.

One final point is worth noting here. The *ip multicast boundary* command is not the same as simply putting an access list on the interface to block the exchange of packets. The actual multicast packets are not the only thing that you want to prevent from crossing this boundary. You also want to prevent PIM from joining any of these multicast distribution trees across the boundary. That is what this command does for you.

23.11.4 See Also

Recipe 23.10; RFC 1918; RFC 2365

Top

Recipe 23.12 Exchanging Multicast Routing Information with MBGP

23.12.1 Problem

You want to exchange multicast routing information between two networks using MBGP.

23.12.2 Solution

Before setting up MBGP, you should set up multicast routing on the Autonomous System Boundary Router (ASBR) and configure it to block multicast traffic that you know is intended only for the local network:

```
Router-ASBR1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router-ASBR1(config)#ip multicast-routing
Router-ASBR1(config)#access-list 15 deny 239.0.0.0 0.255.255.255
Router-ASBR1(config)#access-list 15 deny 224.0.1.39
Router-ASBR1(config)#access-list 15 deny 224.0.1.40
Router-ASBR1(config)#access-list 15 permit any
Router-ASBR1(config)#interface Serial0/0
Router-ASBR1(config-if)#ip multicast boundary 15
Router-ASBR1(config-if)#ip multicast ttl-threshold 64
Router-ASBR1(config-if)#ip pim dense-mode
Router-ASBR1(config-if)#end
Router-ASBR1#
```

Then you need to set up the MBGP configuration:

```
Router-ASBR1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router-ASBR1(config)#router bgp 65530
Router-ASBR1(config-router)#network 10.0.0.0 mask 255.0.0.0
Router-ASBR1(config-router)#neighbor 10.15.32.1 remote-as 65531
Router-ASBR1(config-router)#address-family ipv4 multicast
Router-ASBR1(config-router-af)#neighbor 10.15.32.1 activate
Router-ASBR1(config-router-af)#end
Router-ASBR1#
```

23.12.3 Discussion

Usually when people talk about using BGP, they immediately think of the public Internet. Since most of the Internet is not capable of transmitting multicast traffic (yet), MBGP may not seem immediately

useful. However, BGP has many other uses besides connecting to the Internet. For example, many large networks use it for interconnecting larger corporate divisions for stability, scalability, or administrative reasons. BGP is often used for interconnecting private networks belonging to separate companies that share information in large volume. In any case where you use BGP for interconnecting two networks, it is natural to consider MBGP for sharing required multicast routing information. And there is more and more interest and experimentation in multicast functionality in the public Internet.

However, it's important to remember that MBGP is not actually a multicast routing protocol in the same sense as PIM or DVMRP. It does not do Join or Prune operations to create SPTs, nor does it have a mechanism to find RPs. It merely allows you to transmit routing information that the router can use in calculating the best path back to the source. This is why we have configured PIM-DM on the external interface in the example.

The reason that we have not specified PIM-SM in particular is that doing so implies that there must be an RP external to the AS. This is possible, and increasingly common. But it means that you need a way to discover it. The best way to do this is to use the Multicast Source Discovery Protocol (MSDP), which is described in Recipe 23.13.

The example configuration does several things. First, it uses the same principles demonstrated in Recipe 23.10 and Recipe 23.11 for controlling scope. The external interfaces drop any packets with a TTL value less than or equal to 64, to help prevent internal applications from reaching the adjacent network. And these interfaces are also configured to block all groups with addresses between `239.0.0.0` and `239.255.255.255`, to enforce administratively scoped addressing.

You will also notice that the same access list that enforces this address restriction also blocks two other groups: `224.0.1.39` and `224.0.1.40`. These are used by Cisco's proprietary Auto-RP for discovering RPs within a network. It is a good idea to prevent these groups from crossing network boundaries, whether you are using Auto-RP or not. Otherwise you risk leaking inappropriate RP information from one network into the other. It can cause serious confusion if your network tries to use the RP from an adjacent network for its internal traffic.

In the BGP configuration section, both multicast and unicast traffic use the same network paths. You can also break these up so that you send multicast traffic by a different path than unicast traffic. This is done by simply defining one of the BGP peers for multicast traffic, and leaving the other unmodified so that the router will use it for unicast traffic.

```
Router-ASBR1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router-ASBR1(config)#router bgp 65530
Router-ASBR1(config-router)#network 10.0.0.0 mask 255.0.0.0
Router-ASBR1(config-router)#neighbor 10.15.32.1 remote-as 65531
Router-ASBR1(config-router)#neighbor 10.15.32.2 remote-as 65531
Router-ASBR1(config-router)#address-family ipv4 multicast
Router-ASBR1(config-router-af)#neighbor 10.15.32.1 activate
Router-ASBR1(config-router-af)#end
```

```
Router-ASBR1#
```

Whether you route unicast and multicast traffic through the same or different paths, MBGP allows you to apply AS filtering separately to both kinds of traffic. There is a new route map match clause that you can use specifically to identify multicast routing information:

```
Router-ASBR1(config)#route-map mbgp-test permit 10
Router-ASBR1(config-routemap)#match nlri multicast
```

23.12.4 See Also

Recipe 23.10; Recipe 23.11; Recipe 23.13

Top

Recipe 23.13 Using MSDP to Discover External Sources

23.13.1 Problem

You want to use MSDP to discover information about multicast sources in other ASes.

23.13.2 Solution

The typical way to configure MSDP involves first selecting one of your MBGP routers as the RP for your internal network. Then you set up an MSDP peer relationship with the RP in another AS, which is usually an MBGP peer router in the next domain. The following configuration includes the commands required to configure the router as an RP for the internal network using BSR, as discussed in Recipe 23.2; configuration to prevent local multicast traffic from leaking into the neighboring network, as discussed in Recipe 23.10 and Recipe 23.11; and BGP configuration, as discussed in Recipe 23.12:

```
Router-ASBR1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router-ASBR1(config)#ip multicast-routing
Router-ASBR1(config)#interface Loopback0
Router-ASBR1(config-if)#ip address 192.168.12.1 255.255.255.255
Router-ASBR1(config-if)# ip pim sparse-mode
Router-ASBR1(config-if)#interface FastEthernet0/0
Router-ASBR1(config-if)#ip address 192.168.1.1 255.255.255.0
Router-ASBR1(config-if)#ip pim sparse-mode
Router-ASBR1(config-if)#exit
Router-ASBR1(config)#interface Serial1/0
Router-ASBR1(config-if)#ip address 192.168.2.5 255.255.255.252
Router-ASBR1(config-if)#ip multicast boundary 15
Router-ASBR1(config-if)#ip multicast ttl-threshold 64
Router-ASBR1(config-if)#ip pim sparse-mode
Router-ASBR1(config-if)#exit
Router-ASBR1(config)#ip pim rp-candidate loopback0
Router-ASBR1(config)#ip pim bsr-candidate loopback0 1
Router-ASBR1(config-if)#router bgp 65530
Router-ASBR1(config-router)#network 10.0.0.0 mask 255.0.0.0
Router-ASBR1(config-router)#neighbor 192.168.2.6 remote-as 65531
Router-ASBR1(config-router)#address-family ipv4 multicast
Router-ASBR1(config-router-af)#neighbor 192.168.2.6 activate
Router-ASBR1(config-router-af)#exit
Router-ASBR1(config-router)#exit
Router-ASBR1(config)#ip msdp peer 192.168.2.6
Router-ASBR1(config)#ip msdp sa-request 192.168.2.6
```

```
Router-ASBR1(config)#access-list 15 deny 239.0.0.0 0.255.255.255
Router-ASBR1(config)#access-list 15 deny 224.0.1.39
Router-ASBR1(config)#access-list 15 deny 224.0.1.40
Router-ASBR1(config)#access-list 15 permit any
Router-ASBR1(config)#end
Router-ASBR1#
```

23.13.3 Discussion

This is a rather long example, but most of the information here is discussed in earlier recipes. The important lines are just the two *ip msdp* commands:

```
Router-ASBR1(config)#ip msdp peer 192.168.2.6
Router-ASBR1(config)#ip msdp sa-request 192.168.2.6
```

This tells the router that it is to send Source Active (SA) messages to this peer device whenever it sees new multicast sources, and requests that the peer device do the same. The peer must also be configured with similar commands to ensure that it exchanges information about its multicast sources with this router.

You can configure several MSDP peers, but if you do, one of them should be configured as the default:

```
Router-ASBR1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router-ASBR1(config)#ip msdp peer 192.168.2.6 connect-source Loopback0
Router-ASBR1(config)#ip msdp sa-request 192.168.2.6
Router-ASBR1(config)#ip msdp default-peer 192.168.3.6 connect-source Loopback0
Router-ASBR1(config)#ip msdp sa-request 192.168.3.6
Router-ASBR1(config)#end
Router-ASBR1#
```

Even though MSDP will allow the routers to exchange information about active sources and groups, it does not include the ability to Join and Prune SPTs for different multicast groups. So, for this reason, we have configure PIM-SM on the connection to the peer router.

This multipeer example also shows how to specify a particular source address for this router. In this case, MSDP will use the address of the `Loopback0` interface.

You can also apply filters to both the sources and groups that you want to receive or distribute to another network with the *ip msdp sa-filter* command:

```
Router-ASBR1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router-ASBR1(config)#ip msdp sa-filter in 192.168.2.6 list 101
Router-ASBR1(config)#ip msdp sa-filter out 192.168.2.6 list 102
Router-ASBR1(config)#access-list 101 permit any 225.0.0.0 0.255.255.255
Router-ASBR1(config)#access-list 101 permit 10.0.0.0 0.255.255.255 any
Router-ASBR1(config)#access-list 101 deny any any
Router-ASBR1(config)#end
```

```
Router-ASBR1#
```

23.13.4 See Also

Recipe 23.2; Recipe 23.3; Recipe 23.10; Recipe 23.11; Recipe 23.12

Top

Recipe 23.14 Converting Broadcasts to Multicasts

23.14.1 Problem

You have a broadcast-based application that you want to treat as multicast so that it can cross the network.

23.14.2 Solution

Cisco has a special feature called an IP Multicast Helper, which you can use to convert broadcast packets to multicast packets. Then you can use PIM to send these packets throughout the network. At the last-hop routers you can then convert the multicast packets back to broadcast. This is useful for older broadcast-based applications that do not support multicast transmission.

Router1 is the first-hop router, or the one closest to the broadcast source, which is on the interface `FastEthernet0/0`. This converts broadcast packets with UDP port `3535` received on this interface into multicast packets in group `239.3.5.35`:

```
Router1#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router1(config)#ip multicast-routing
Router1(config)#access-list 115 permit any any udp 3535
Router1(config)#access-list 115 deny any any udp
Router1(config)#interface FastEthernet0/0
Router1(config-if)#ip directed broadcast
Router1(config-if)#ip multicast helper-map broadcast 239.3.5.35 115
Router1(config)#ip pim sparse-dense-mode
Router1(config-if)#exit
Router1(config)#ip forward-protocol udp 3535
Router1(config)#end
Router1#
```

The last-hop router's configuration is similar, except that it must be configured to turn multicast packets for this group back into broadcasts:

```
Router2#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Router2(config)#ip multicast-routing
Router2(config)#access-list 115 permit any any udp 3535
Router2(config)#access-list 115 deny any any udp
Router2(config)#interface Ethernet0
```

```
Router2(config-if)#ip address 192.168.9.1 255.255.255.0
Router2(config-if)#ip directed broadcast
Router2(config-if)#ip multicast helper-map 239.3.5.35 192.168.9.255 115
Router2(config-if)#ip pim sparse-dense-mode
Router2(config)#ip igmp join-group 239.3.5.35
Router2(config-if)#exit
Router2(config)#ip forward-protocol udp 3535
Router2(config)#end
Router2#
```

23.14.3 Discussion

Before explaining this recipe in any detail, we would like to stress that the multicast helper feature should only be used as a temporary measure until a proper multicast application can be found. It tends to consume a lot of the router's CPU resources. And it can be difficult to troubleshoot application problems if the router is rewriting the packets. It is always preferable to use a native multicast application if possible.

The most important lines in this example are the *ip multicast helper-map* commands that are applied on the two routers. The command on Router1 converts broadcast to multicasts with a group address of `239.3.5.35`:

```
Router1(config-if)#ip multicast helper-map broadcast 239.3.5.35 115
```

Then Router2 converts this group to the network broadcast address `192.168.9.255` of the destination network:

```
Router2(config-if)#ip multicast helper-map 239.3.5.35 192.168.9.255 115
```

End devices on the destination network can now receive the broadcast as if a device on this same segment had sent it.

This example doesn't convert all broadcasts received on the FastEthernet port of Router1 to multicasts. It first applies the access list number 115 to broadcasts that it receives. This picks out a single UDP port, number `3535`, for conversion. If you wanted to convert other broadcasts received on this port as well, it is simply a matter of opening up this access list.

There are three additional commands in these configuration examples that are critical to the broadcast multicast conversion working properly. First is the *ip forward-protocol* command. The multicast conversion process is done in the router's CPU, so it cannot be fast switched. By default, the router will ignore all broadcasts except for a few important UDP ports such as NetBIOS. This command forces the routers to see the broadcast packets so that it can decide whether to process them.

Second, and related to this, is the *ip directed-broadcast* command. A directed broadcast is one that is sent to a particular network or group of networks. So, for example, Router2 in the recipe turns the multicast packet into the directed broadcast with an address of `192.168.9.255`. By default, a Cisco router will

drop all incoming directed broadcasts. So this needs to be enabled on both routers. However, this command can be dangerous on public network segments. There are several well-known DoS attacks, most notably the *smurf* and *fraggle* attacks, that take advantage of directed broadcasts.

Finally, we have included a static IGMP Join on the destination interface. Recall that in Recipe 23.6 we used this command when the devices on this interface required a group but didn't implement IGMP properly. In this case, the devices on the segment don't even know that there is a multicast group to join. So we can use this command to ensure that this router receives the group. Otherwise the multicast packets will never reach Router2.

You should be careful when using multicast helper commands in a network that uses TTL scoping. Cisco doesn't provide a way to adjust the initial TTL setting on these multicast packets. So you may need to set up address-based boundaries (discussed in Recipe 23.11) to prevent these artificial multicasts from leaking out of the network.

One last point on the subject of broadcast to multicast conversion might be useful in some rare cases. If you have an application that is capable of sending multicasts, but have end devices that can only receive broadcasts, you might be able to use only the last-hop (Router2) configuration to get the packets to the receiving devices. Similarly, if you have a server that can only broadcast, but end devices that understand multicasts, you could conceivably use just the first-hop (Router1) configuration. However, it is unlikely that you will encounter such strange applications in any production network.

23.14.4 See Also

Recipe 23.6; Recipe 23.11

Top

Recipe 23.15 Showing Multicast Status

23.15.1 Problem

You want to view the current status of multicast protocols on your router.

23.15.2 Solution

There are several useful commands for checking the status of multicast configuration and protocols. You can see what multicast routes pass through a router with the EXEC command:

```
Router#show ip mroute
```

There are two useful variants of this command. The first reports on forwarding statistics for each multicast group:

```
Router#show ip mroute count
```

The second reports only on the groups that are currently active:

```
Router#show ip mroute active
```

You can look at statistics on group membership using the following command:

```
Router#show ip igmp groups
```

Use the *interface* keyword to look at the IGMP information in more detail:

```
Router#show ip igmp interface
```

There are four useful commands for viewing PIM information. The first shows information about PIM neighbor relationships:

```
Router#show ip pim neighbor
```

The second command shows information about the PIM configuration on different interfaces:

```
Router#show ip pim interface
```

This command shows information about PIM-SM RPs:

```
Router#show ip pim rp
```

And, if you are using the Bootstrap Router (PIM Version 2) technique for distributing RP information, you will want to use this command:

```
Router#show ip pim bsr-router
PIMv2 Bootstrap information
This system is the Bootstrap Router (BSR)
  BSR address: 172.17.254.5 (?)
  Uptime:      00:06:37, BSR Priority: 0, Hash mask length: 1
  Next bootstrap message in 00:00:22

Next Cand_RP_advertisement in 00:00:15
  RP: 172.17.254.5(Loopback0)
Router#
```

There are several commands that allow you to look at MSDP functions:

```
Router#show ip msdp count

Router#show ip msdp peer 192.168.201.15

Router#show ip msdp summary
```

You can look at details of the Reverse Path Forwarding trees using two useful commands, *show ip rpf* and *mstat*:

```
Router#show ip rpf 192.168.3.2

Router#mstat 192.168.3.2 239.5.5.55
```

23.15.3 Discussion

The *show ip mroute* command gives information on multicast routing. It shows which interfaces the router will use to forward packets belonging to different groups, and it also shows where the sources or RPs for these groups are:

```
Router#show ip mroute
IP Multicast Routing Table
Flags: D - Dense, S - Sparse, C - Connected, L - Local, P - Pruned
       R - RP-bit set, F - Register flag, T - SPT-bit set, J - Join SPT
Timers: Uptime/Expires
Interface state: Interface, Next-Hop or VCD, State/Mode

(*, 224.0.1.40), 03:29:10/00:00:00, RP 0.0.0.0, flags: DJCL
  Incoming interface: Null, RPF nbr 0.0.0.0
  Outgoing interface list:
     Ethernet1, Forward/Sparse-Dense, 03:29:10/00:00:00

(*, 239.5.5.55), 5d06h/00:02:59, RP 0.0.0.0, flags: DJC
  Incoming interface: Null, RPF nbr 0.0.0.0
  Outgoing interface list:
```

```
    Ethernet0, Forward/Sparse-Dense, 00:04:23/00:00:00
    Ethernet1, Forward/Sparse-Dense, 03:29:08/00:00:00

(192.168.5.2/32, 239.5.5.55), 00:00:50/00:02:09, flags: CT
  Incoming interface: Ethernet1, RPF nbr 0.0.0.0
  Outgoing interface list:
    Ethernet0, Forward/Sparse-Dense, 00:00:50/00:00:00
```

In this example, the router belongs to multicast trees for two groups, `224.0.1.40` and `239.5.5.55`. The first of these groups is what Cisco routers use for the Auto-RP procedure to locate an RP. In this case, there is no source for this group, but there is a member on the segment Ethernet1. This group member is another router that is also looking for the RP.

The second group has two entries. The first is the general (*,*G*) entry. The second is a specific (*S*,*G*) entry that shows the only known source for this multicast group, `192.168.5.2`. This specific entry indicates that this source is on the interface `Ethernet1`, and that there are members on interface Ethernet0.

The flags at the end of each group entry give useful information about the forwarding. Both of the general (*,*G*) entries include the flags "D" and "J", which indicates that the router is using dense-mode forwarding, and that it has joined a Shortest Path Tree. This is important because it says that although this router is configured for sparse-dense mode forwarding, it has not yet found an RP, so it is using dense-mode forwarding.

The "C" flag, which you can see in all of the multicast routing entries, is for connected networks. This is most useful in the last entry because it shows that the multicast source, `192.168.5.2`, is on a directly connected network segment. Since both the source and all of the group members are on interfaces on the same router, there is no need for it to try to join a tree connecting it to other routers.

The other interesting flag in this example is the "L" flag, which indicates that the router itself is a member of this group. This is true for the `224.0.1.40` group, because the router is hoping to discover the RP by listening to this group.

With the *count* keyword, this command gives statistics on multicast usage:

```
Router#show ip mroute count
IP Multicast Statistics
3 routes using 1776 bytes of memory
2 groups, 0.5 average sources per group
Forwarding Counts: Pkt Count/Pkts per second/Avg Pkt Size/Kilobits per second
Other counts: Total/RPF failed/Other drops(OIF-null, rate-limit etc)

Group: 224.0.1.40, Source count: 0, Group pkt count: 0

Group: 239.5.5.55, Source count: 1, Group pkt count: 1
  Source: 192.168.5.2/32, Forwarding: 1/0/100/0, Other: 1/0/0
```

The entry for `239.5.5.55` shows that there is only one source. It also shows that there has been only one packet associated with this group for all sources. The detail below this line breaks out the packet statistics for each source. The fields work as follows:

```
Source: 192.168.5.2/32, Forwarding: 1/0/100/0, Other: 1/0/0
```

There are four fields separated by slashes following the word "Forwarding." These represent the total number of packets that have been forwarded, the current forwarding rate in packets per second, the average packet size, and the forwarding rate in kilobits per second. The three fields after the word "Other" are the total number of packets received, followed by the number of packets that had to be dropped for Reverse Path Forwarding failures, and the number of drops for all other reasons. Note that the total number of packets forwarded, from the first number after "Forwarding," plus the last two fields after "Other" should add up to the first field after "Other." These last two fields simply give an indication of when packets are dropped and why.

The *active* keyword shows only those sources that are currently sending multicast traffic at a rate greater than 4Kbps:

```
Router#show ip mroute active
Active IP Multicast Sources - sending >= 4 kbps

Group: 239.5.5.55, (?)
   Source: 192.168.5.2 (?)
     Rate: 6 pps/41 kbps(1sec), 14 kbps(last 15 secs), 1 kbps(life avg)
   Source: 192.168.254.5 (?)
     Rate: 2 pps/24 kbps(1sec), 12 kbps(last 15 secs), 1 kbps(life avg)
Router#
```

This command can be deceptive because many multicast applications operate significantly slower than the minimum 4Kbps rate. The sources for those slower applications do not appear. However, this can be useful for higher bandwidth multimedia type applications. The output shows the rates in both packets per second (pps) and kilobits per second (Kbps) averaged over the last second, as well as the rates in Kbps for the last 15 seconds and averaged over the entire life of this source.

In this particular case, there are actually three sources for this multicast application, but one is below the threshold. You can see them all using the following command:

```
Router#show ip mroute 239.5.5.55 count
IP Multicast Statistics
12 routes using 4908 bytes of memory
5 groups, 1.40 average sources per group
Forwarding Counts: Pkt Count/Pkts per second/Avg Pkt Size/Kilobits per second
Other counts: Total/RPF failed/Other drops(OIF-null, rate-limit etc)

Group: 239.5.5.55, Source count: 3, Group pkt count: 2319
  RP-tree: Forwarding: 0/0/0/0, Other: 0/0/0
  Source: 192.168.3.2/32, Forwarding: 65/0/76/0, Other: 65/0/0
  Source: 192.168.5.2/32, Forwarding: 1127/0/564/0, Other: 1128/1/0
```

```
  Source: 192.168.254.5/32, Forwarding: 1127/0/564/0, Other: 1127/0/0
Router#
```

To look at multicast group membership on the router, use the following command:

```
Router# show ip igmp groups
Group Address    Interface              Uptime    Expires   Last Reporter
224.0.1.39       TokenRing0             01:17:44  00:02:59  192.168.3.2
224.0.1.40       Ethernet1              16:02:35  never     192.168.5.1
239.5.5.55       Ethernet0              00:22:07  00:02:50  192.168.1.104
239.5.5.55       TokenRing0             16:02:26  00:02:53  192.168.3.2
224.0.1.1        TokenRing0             16:02:26  00:02:51  192.168.3.2
Router#
```

This tells you where there are members for each of the active groups. The "Last Reporter" column tells you the last known group member for this group. The router will periodically query the segment to make sure that there are still active members. The "Uptime" column indicates how long this group has had members on this segment, and the "Expires" column shows when this group will be removed if there are no further membership reports. Note that one of the entries, `224.0.1.40`, will never expire. This is because the router itself is a member of this group; it is used for the Auto-RP process.

With the *interface* keyword, the *show ip igmp* command gives details on how IGMP is implemented on a particular interface. It tells you information about IGMP query periods and what router is the PIM Designated Router (DR) for the segment. In the following example, this router is the DR for the segment. When there is more than one router on a segment, one of them must claim the role of DR, or every multicast packet will appear twice. So this is a useful way of figuring out which router is the DR:

```
Router# show ip igmp interface Ethernet0
Ethernet0 is up, line protocol is up
  Internet address is 192.168.1.201/24
  IGMP is enabled on interface
  Current IGMP version is 2
  CGMP is disabled on interface
  IGMP query interval is 60 seconds
  IGMP querier timeout is 120 seconds
  IGMP max query response time is 10 seconds
  Inbound IGMP access group is not set
  IGMP activity: 2 joins, 0 leaves
  Multicast routing is enabled on interface
  Multicast TTL threshold is 15
  Multicast designated router (DR) is 192.168.1.201 (this system)
  IGMP querying router is 192.168.1.201 (this system)
  No multicast groups joined
```

Another useful piece of information in this example is the *multicast TTL threshold* indicator. A TTL threshold of 15 has been set on this interface to prevent local multicast traffic from reaching this segment. One of the most common problems with multicast networks comes from incorrectly setting multicast TTL thresholds. This can either allow multicast traffic to leak out of the appropriate network region, or

cause a router to drop this traffic prematurely.

You can look at a router's PIM neighbor table as follows, for IOS Version 11.x:

```
Router#show ip pim neighbor
PIM Neighbor Table
Neighbor Address  Interface          Uptime    Expires   Mode
192.168.5.2       Ethernet1          16:22:46  00:01:23  Sparse-Dense (DR)
192.168.3.2       TokenRing0         16:22:16  00:01:05  Sparse-Dense (DR)
Router#
```

This shows not only what the PIM neighbor routers are, but also what PIM mode they are using (sparse-dense for both of the routers in this example), how long the neighbor relationship has existed, and when the router will delete this neighbor from its table if it doesn't hear from it again. The very last field in both lines indicates that this router is the DR for both of these network segments. If you look at the same information on either of these neighbor routers, you will see that neither of them have this DR indication.

The format of this output changed slightly between IOS Versions 11.x and 12.x. In Version 12.x, it includes PIM version information:

```
Router#show ip pim neighbor
PIM Neighbor Table
Neighbor Address  Interface          Uptime    Expires   Ver  Mode
192.168.5.1       Ethernet0          16:33:41  00:01:02  v1   Dense
192.168.254.6     Serial0            16:34:23  00:01:38  v2
Router#
```

As you can see, one of the neighbors is using PIM Version 1. This actually highlights the interoperability of the different PIM versions. You can easily build a hybrid network using both old and new routers.

The *show ip pim interface* command looks like this:

```
Router#show ip pim interface

Address          Interface          Version/Mode    Nbr    Query     DR
                                                    Count  Intvl
192.168.5.2      Ethernet0          v2/Sparse-Dense  1     30      192.168.5.2
192.168.254.5    Serial0            v2/Sparse-Dense  1     30      0.0.0.0
Router#
```

Note that there are slight differences between the output of this command and the previous one. In particular, the *interface* command indicates that the `Ethernet0` port is configured for PIM Version 2 (the default) using sparse-dense mode. However, the previous *neighbor* command shows that this port is actually using dense mode and PIM Version 1 for compatibility with the older router.

When using PIM-SM, you can see information about the RPs using this command:

```
Router#show ip pim rp
```

```
Group: 239.255.255.250, RP: 192.168.3.2, v2, v1, uptime 00:00:43, expires 00:02:16
Group: 239.5.5.55, RP: 192.168.3.2, v2, v1, uptime 00:00:42, expires 00:02:17
Group: 224.0.1.1, RP: 192.168.3.2, v2, v1, uptime 00:00:42, expires 00:02:17
Router#
```

Note that you can have different RPs for different groups. So the output of this command shows the RP separately for each group. Of course, in this example there is only one RP for the entire network.

If you are using a BSR to distribute RP information, as in Recipe 23.2, you can look at the BSR status using the *bsr-router* keyword:

```
Router#show ip pim bsr-router
PIMv2 Bootstrap information
This system is the Bootstrap Router (BSR)
  BSR address: 172.17.254.5 (?)
  Uptime:      00:06:37, BSR Priority: 0, Hash mask length: 1
  Next bootstrap message in 00:00:22

Next Cand_RP_advertisement in 00:00:15
  RP: 172.17.254.5(Loopback0)
Router#
```

In this example, the router itself claims to be the BSR for the network. This means simply that it is responsible for distributing RP information to the network. This command also shows what the next RP that this router intends to advertise will be, and when it will send out the next advertisement.

If you are running MSDP to distribute multicast source and RP information between networks, you will want to use the various *show ip msdp* commands. The *count* keyword allows you to see gross statistics for all of the configured MSDP peers:

```
Router#show ip msdp count
SA State per Peer Counters, <Peer>: <# SA learned>
    192.168.199.15: 0
    192.168.201.15: 0

SA State per ASN Counters, <asn>: <# sources>/<# groups>
    Total entries: 0
```

In this case the peers are configured, but no sources or groups have been learned.

You can look at one peer in more detail with the *peer* keyword:

```
Router#show ip msdp peer 192.168.201.15
MSDP Peer 192.168.201.15 (?), AS ?
Description:
  Connection status:
    State: Down, Resets: 0, Connection source: none configured
    Uptime(Downtime): 00:13:28, Messages sent/received: 0/0
    Output messages discarded: 0
    Connection and counters cleared 00:13:28 ago
```

```
  SA Filtering:
    Input (S,G) filter: none, route-map: none
    Input RP filter: none, route-map: none
    Output (S,G) filter: none, route-map: none
    Output RP filter: none, route-map: none
  SA-Requests:
    Input filter: none
    Sending SA-Requests to peer: disabled
  Peer ttl threshold: 0
  SAs learned from this peer: 0
  Input queue size: 0, Output queue size: 0
```

And the *summary* keyword shows general information about all of the MSDP peers:

```
Router#show ip msdp summary
MSDP Peer Status Summary
Peer Address     AS    State    Uptime/  Reset SA    Peer Name
                                Downtime Count Count
192.168.199.15   65531 Down     00:15:41 0     0     ?
192.168.201.15   ?     Down     00:15:30 0     0     ?
Router#
```

The last version shows a critical piece of information. These MSDP peers are currently unreachable, so no routing information is being exchanged. In this case, the peers have been down for over 15 minutes, allowing you to isolate exactly when the problem occurred.

The last two commands show details on the actual multicast trees. The first, *show ip rpf*, shows information about how the router would build an RPF tree for a specified source. Note that this does not actually require that this source must exist, or that it be currently sending multicast packets. However, the interface leading to this source must be configured for multicast forwarding:

```
Router#show ip rpf 192.168.3.2
RPF information for ? (192.168.3.2)
  RPF interface: Ethernet0
  RPF neighbor: ? (192.168.5.1)
  RPF route/mask: 192.168.3.0/255.255.255.0
  RPF type: unicast
Router#
```

If there are two or more equal cost paths to the same destination, the router simply selects the one with the highest IP address:

```
Router#show ip route 192.168.4.29
Routing entry for 192.168.4.28/30
  Known via "eigrp 65530", distance 90, metric 297372416, type internal
  Redistributing via eigrp 65530
  Last update from 192.168.4.26 on Tunnel6, 00:07:25 ago
  Routing Descriptor Blocks:
  * 192.168.4.22, from 192.168.4.22, 00:07:25 ago, via Tunnel5
      Route metric is 297372416, traffic share count is 1
```

```
        Total delay is 505000 microseconds, minimum bandwidth is 9 Kbit
        Reliability 255/255, minimum MTU 1514 bytes
       Loading 1/255, Hops 1
     192.168.4.26, from 192.168.4.26, 00:07:25 ago, via Tunnel6
       Route metric is 297372416, traffic share count is 1
        Total delay is 505000 microseconds, minimum bandwidth is 9 Kbit
        Reliability 255/255, minimum MTU 1514 bytes
       Loading 1/255, Hops 1

Router#show ip rpf 192.168.4.29
RPF information for ? (192.168.4.29)
  RPF interface: Tunnel6
  RPF neighbor: ? (192.168.4.26)
  RPF route/mask: 192.168.4.28/255.255.255.252
  RPF type: unicast
Router#
```

This is helpful when debugging multicast routing issues because it tells you, for example, that this route uses the unicast routing table. If you had intended for this routing information to come from some other source, such as DVMRP or MBGP, it tells you that there is probably an administrative distance problem. Here is how the output looks for another source that has a static multicast route:

```
Router#show ip rpf 192.168.55.5
RPF information for ? (192.168.55.5)
  RPF interface: Serial0
  RPF neighbor: ? (192.168.254.6)
  RPF route/mask: 192.168.55.5/255.255.255.0
  RPF type: static mroute
Router#
```

For tracing the multicast trees for real sources, you can use the *mstat* command to get much more detail:

```
Router>mstat 192.168.3.2 239.5.5.55
Type escape sequence to abort.
Mtrace from 192.168.3.2 to 192.168.5.2 via group 239.5.5.55
From source (?) to destination (?)
Waiting to accumulate statistics......
Results after 10 seconds:

  Source          Response Dest    Packet Statistics For     Only For Traffic
192.168.3.2     192.168.5.2        All Multicast Traffic     From 192.168.3.2
     |        _ _/  rtt 7    ms     Lost/Sent = Pct  Rate      To 239.5.5.55
     v      /      hop 7    ms     ---------------------     --------------------
192.168.3.1
192.168.5.1     ?
     |     ^      ttl   0
     v     |      hop 0    ms     -2/0 = --%      0 pps     0/0 = --%  0 pps
192.168.5.2     ?
     |      \_ _   ttl   1
     v         \  hop 0    ms        0          0 pps           0    0 pps
```

```
192.168.5.2      192.168.5.2
  Receiver       Query Source

Router>
```

This command tells the router to actually probe the RPF tree using a similar technique to that used by the *traceroute* program. It presents the results starting at the top with the source and tracing down to the receiver (which is the router itself) at the bottom, showing all of the intermediate router hops. There is a lot of useful information in this output. It tells you, for example, the minimum source TTL value required to reach each point in the network. And it also shows the multicast forwarding packet statistics at each hop for both this group and for all multicast traffic. You can use this to determine if you are losing multicast traffic due to congestion at some point in your network.

Here is another interesting *mstat* output in which the router discovers a TTL boundary:

```
Router#mstat 192.168.1.201 239.5.5.55
Type escape sequence to abort.
Mtrace from 192.168.1.201 to 192.168.254.5 via group 239.5.5.55
From source (?) to destination (?)
Waiting to accumulate statistics......
Results after 10 seconds:

  Source         Response Dest    Packet Statistics For     Only For Traffic
0.0.0.0          192.168.254.5    All Multicast Traffic     From 192.168.1.201
     |       _ _/  rtt 3    ms     Lost/Sent = Pct  Rate      To 239.5.5.55
     v      /     hop 3    ms     ---------------------     -----------------
0.0.0.0
192.168.254.6    ? Hit scope boundary
     |     ^      ttl   64
     v     |      hop 0    ms     -2/0 = --%      0 pps     0/0 = --%  0 pps
192.168.254.5    ?
     |      \_ _   ttl   65
     v         \  hop 0    ms        0         0 pps           0    0 pps
192.168.254.5    192.168.254.5
  Receiver       Query Source

Router#
```

Note that at the `192.168.254.6` hop, *mstat* discovers that there is a TTL boundary with a threshold value of 64. So, you can see that the minimum TTL required to reach the final destination is 65. This is an extremely useful command for isolating multicast routing problems.

23.15.4 See Also

Recipe 23.2; Recipe 23.3; Recipe 23.13

Top

Recipe 23.16 Debugging Multicast Routing

23.16.1 Problem

You want to use debug functions to isolate a problem with multicast forwarding.

23.16.2 Solution

Cisco routers have several useful debug features that you can use to isolate multicast problems. The first is a general command that shows how the router maintains its multicast routing tables when it hears from sources and group members:

```
Router#debug ip mrouting
```

You can watch the actual multicast packets for a particular group using the following command:

```
Router#debug ip mpacket 239.5.5.55
```

The other commonly useful multicast debug command looks at IGMP information:

```
Router#debug ip igmp
```

23.16.3 Discussion

As with all debugging commands, you need to be extremely careful because sometimes the sheer volume of the output can overwhelm the router. It is usually wise to try these commands one at a time, and disable all debugging with the command *undebug all* before trying the next command.

The first debug command, *debug ip mrouting* , shows how the router creates, updates, and deletes multicast routing information:

```
Router#terminal monitor
Router#debug ip mrouting
IP multicast routing debugging is on
Router#
17:20:27: MRT: Create (192.168.5.1/32, 239.5.5.55), RPF Ethernet0/0.0.0.0, PC
0x33A89D8
17:20:43: MRT: Update (*, 224.0.1.40), RPF Null, PC 0x339F96C
17:20:49: MRT: Delete (192.168.3.2/32, 224.0.1.39), PC 0x33AB26A
17:21:43: MRT: Update (*, 224.0.1.40), RPF Null, PC 0x339F96C
17:21:49: MRT: Create (192.168.3.2/32, 224.0.1.39), RPF Ethernet0/192.168.5.1, PC
0x33A89D8
17:22:13: MRT: Delete (*, 224.0.1.1), PC 0x33AB26A
17:22:24: MRT: Create (*, 224.0.1.1), RPF Null, PC 0x33A8890
```

```
17:22:46: MRT: Update (*, 224.0.1.40), RPF Null, PC 0x339F96C
17:22:46: MRT: Update (*, 224.0.1.40), RPF Null, PC 0x339F96C
```

In this example, the first line creates a group entry for `239.5.5.55` with the source `192.168.5.1` in response to receiving a multicast packet from this source. You can also see a number of entries here for the Auto-RP groups, `224.0.1.39` and `224.0.1.39` . These are the result of routers chatting amongst themselves to ensure that a stable RP exists for the network.

The next command, *debug ip mpacket* , shows individual multicast packets. Looking at packet-level debug traces is always particularly dangerous because of the possibility of overwhelming the router. In this example, we have asked the router to show only the group `239.5.5.55` :

```
Router#debug ip mpacket 239.5.5.55
IP multicast packets debugging is on for group 239.5.5.55
May 10 16:18:40.870: IP: s=192.168.5.2 (Ethernet1) d=239.5.5.55 (TokenRing0) len
 114, mforward
May 10 16:18:40.874: IP: s=192.168.5.2 (Ethernet1) d=239.5.5.55 (Ethernet0) len
114, mforward
May 10 16:18:40.878: IP: s=192.168.5.2 (TokenRing0) d=239.5.5.55 len 122, not RP
F interface
May 10 16:18:40.890: IP: s=192.168.254.5 (TokenRing0) d=239.5.5.55 (Ethernet1) l
en 122, mforward
May 10 16:18:40.890: IP: s=192.168.254.5 (TokenRing0) d=239.5.5.55 (Ethernet0) l
en 122, mforward
```

As you can see, a packet was received on interface `Ethernet1` with the source address of `192.168.5.2` for this group. The router immediately turned around and forwarded this packet to the `TokenRing0` and `Ethernet0` interfaces. A short time later, it received another packet for this group with the same source address from `TokenRing0` . However it didn't forward this packet along because it was not received on the RPF interface. That is, the router looked in its routing table, realized that this was not the way that it should have received this packet, and dropped it to avoid loops.

The router then receives a multicast packet for this same group on interface `TokenRing0` , but this time it has a source address of `192.168.254.5` . It forwards this packet to both `Ethernet1` and `Ethernet0` .

It is important to note that this command tells you nothing about group membership. If devices Join or Leave this group, you will not see them this way. To do that, you need to look at the output of *debug ip igmp* :

```
Router#debug ip igmp
IGMP debugging is on
17:34:17: IGMP: Send v2 Query on Ethernet0 to 224.0.0.1
17:34:18: IGMP: Send v2 Query on Ethernet1 to 224.0.0.1
17:34:18: IGMP: Set report delay time to 8.6 seconds for 224.0.1.40 on Ethernet1
17:34:18: IGMP: Send v2 Query on TokenRing0 to 224.0.0.1
17:34:19: IGMP: Received v2 Report from 192.168.1.104 (Ethernet0) for 239.5.5.55
17:34:21: IGMP: Received v2 Report from 192.168.5.2 (Ethernet1) or 224.0.1.40
17:34:21: IGMP: Cancel report for 224.0.1.40 on Ethernet1
```

```
17:34:23: IGMP: Received v2 Report from 192.168.3.2 (TokenRing0) for 239.5.5.55
17:34:24: IGMP: Received v2 Report from 192.168.3.2 (TokenRing0) for 224.0.1.39
17:34:25: IGMP: Received v2 Report from 192.168.3.2 (TokenRing0) for 224.0.1.1
17:34:26: IGMP: Received v2 Report from 192.168.1.100 (Ethernet0) for 239.255.255.250
17:34:27: IGMP: Received v2 Report from 192.168.1.104 (Ethernet0) for 239.5.5.55
17:34:32: IGMP: Received v2 Report from 192.168.1.104 (Ethernet0) for 239.5.5.55
```

This debug trace shows a couple of interesting events buried in a whole lot of router-to-router multicast chatter, so you have to look carefully at the group addresses to make sure that you're seeing the interesting data. The group `224.0.0.1` is the *all systems* group, which is used for local segment chatter. It is rarely interesting for multicast routing, because it is intended to be purely local, and all multicast-capable devices are always members. The groups `224.0.1.39` and `224.0.1.40` are used by the Auto-RP protocol to allow routers to share information about the PIM-SM RPs for the network.

So, unless you are trying to debug an Auto-RP problem, the only really interesting group information in this trace is for `239.5.5.55` and `239.255.255.250` . The device `192.168.1.104` has joined the first group, and sends several IGMP report packets to make sure that it has joined successfully. The device `192.168.1.100` , on the other hand, has been a member of `239.255.255.250` for some time and sends only a single packet to ensure that it will continue to receive this group.

A short time later, the device `192.168.1.104` leaves the group `239.5.5.55` , as shown in the following trace:

```
17:34:54: IGMP: Received Leave from 192.168.1.104 (Ethernet0) for 239.5.5.55
17:34:54: IGMP: Send v2 Query on Ethernet0 to 239.5.5.55
17:34:55: IGMP: Send v2 Query on Ethernet0 to 239.5.5.55
17:34:57: IGMP: Deleting 239.5.5.55 on Ethernet0
```

Since this was the last known member of this group on this segment, the router responds to the IGMP Leave message with an IGMP query. It tries twice to see if there are still any other devices interested in continuing to see this group, then deletes it.

Top

Appendix A. External Software Packages

This appendix discusses several of the external software packages discussed throughout the book. Because this is primarily a Cisco book, and we have not focused on any particular software products, this section is restricted to freely distributed software. There are also commercial products that fulfill the same functions as some of these packages (particularly for SNMP) that you may prefer to use.

Top

A.1 Perl

According to the Perl web site:

> Perl is a high-level programming language with an eclectic heritage written by Larry Wall and a cast of thousands. It derives from the ubiquitous C programming language and to a lesser extent from sed, awk, the Unix shell, and at least a dozen other tools and languages. Perl's process, file, and text manipulation facilities make it particularly well-suited for tasks involving quick prototyping, system utilities, software tools, system management tasks, database access, graphical programming, networking, and world wide web programming.

Many of the scripts written in Perl these days tend to involve dynamically generating web pages. But all of the scripts in this book use Perl at the command line of either a Unix or Windows computer.

We frequently use Perl for scripting network administration functions because it is an extremely powerful and flexible language, particularly for requirements involving pattern matching. This makes it perfect for scanning log files, as well as for spawning dynamic queries and formatting the output into a useful report.

Perl is available for both Unix and Windows systems. This is important because, while the engineers who run most of the world's larger networks use Unix, smaller organizations frequently don't have any Unix expertise. So it is not uncommon to see Windows computers managing smaller networks.

Perl's free and open distribution policy means that there is usually a good port available, even if you use a different system. And, most important for organizations on tight budgets, it's free.

The scripts in this book were written and tested with a variety of different releases of Perl Version 5. However, we deliberately wrote the scripts to be as portable as possible, so they should run without alteration in most versions of the language.

The official Perl web page is http://www.perl.com/. This site has a wealth of information to help people who program in Perl, including many helpful ideas for beginners.

You can download the most recent source code for Perl from this web site, which also has compiled versions for a variety of platforms. The following URL will direct you to the download area: http://www.perl.com/pub/a/language/info/software.html.

The Perl web site also has extensive documentation that is quite well-written and easy to follow at http://www.perl.com/pub/q/documentation.

There are also several excellent books on Perl that you may find helpful. Programming Perl, by Larry

Wall, Tom Christiansen and Jon Orwant (O'Reilly) is an excellent introduction to the language and its features. We also recommend Perl In a Nutshell, by Ellen Siever, Stephen Spainhour and Nathan Patwardhan (O'Reilly). And, if you are interested in seeing some of the other things that you can do with this language, have a look at Perl Cookbook, by Tom Christiansen and Nathan Torkington (O'Reilly).

Top

A.2 Expect

Expect is another scripting language that helps solve a different type of problem. Where Perl's strength is in pattern matching, Expect provides a way to automate interactive applications. We usually use Expect to imitate user sessions on a router to automate command-line tasks.

The Expect program is able to send one or more lines of output (such as router commands) and capture the results. It can also react to whatever the router sends it in return. This could be as simple as sending a user ID and waiting for the password prompt, or you could use this feature to check for various error conditions and react appropriately.

We often write scripts in Expect to automate boring, repetitive tasks. Computers are good at these tasks; people aren't. People make typos and get bored, or blink and miss key pieces of information. Also, because Expect can react immediately to the router's responses, the script can generally execute a series of commands very quickly. We think it's better to spend our time doing something productive while the computer is logging into all of our routers to do *show* commands. Expect lets us do this.

Expect is free to download, distribute, and use for any purpose. There are both Unix and Windows versions, and there are even companies doing commercial support for Expect. We wrote and tested all of the scripts in this book using Expect Version 5.31.2 on a Unix platform.

You can download Expect from the official web page at http://expect.nist.gov/. This site also has useful documentation and example scripts. For more information, we highly recommend Exploring Expect, by Don Libes (O'Reilly).

Top

A.3 NET-SNMP

NET-SNMP is a free open source SNMP package that is based on the earlier UCD-SNMP and CMU-SNMP packages developed at University of California at Davis, and Carnegie Mellon University, respectively. The current version supports SNMP Versions 1, 2c, and 3. It is available for both Windows and Unix platforms.

This package includes a complete suite of SNMP programs. It includes SNMP agent and server software, as well as the command-line utilities needed for interacting with SNMP devices to extract information or change settings that we used in our scripts. In fact, we used only a small subset of the NET-SNMP suite of applications in this book.

We wrote and tested all of the scripts in this book using NET-SNMP Version 4.2.

The official NET-SNMP web page is http://www.net-snmp.org/, which is mirrored at http://net-snmp.sourceforge.net/. This site contains documentation and other useful information about the package. You can download the software via FTP from ftp://ftp.net-snmp.org/pub/sourceforge/net-snmp/.

Unfortunately, we are not aware of any books written specifically about NET-SNMP. However, Essential SNMP, by Douglas Mauro and Kevin Schmidt (O'Reilly), does a good job of covering SNMP in general, and includes some discussion of NET-SNMP as well.

Top

A.4 PuTTY

PuTTY is a free implementation of the Telnet and SSH protocols for Windows. Its current version supports Telnet, SSHv1, SSHv2, Secure copy, Secure FTP, and*rlogin*. A client-only version of each protocol is available for the Windows platform.

PuTTY boasts an impressive set of features. Its SSH client is robust and feature-rich, and the Telnet support is far superior to the standard Telnet client that ships with Windows.

The official PuTTY web site is http://www.chiark.greenend.org.uk/~sgtatham/putty/. This site contains documentation and other useful information about the package. You can download the software via FTP from http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html.

Top

A.5 OpenSSH

OpenSSH is a free version of the SSH protocol suite for Unix and Unix-like systems. Its current version supports SSHv1, SSHv2, Secure copy, and Secure FTP. Not only does OpenSSH provide SSH clients, it also includes the server-side software. OpenSSH does not currently support Windows-based systems.

OpenSSH initially started out as an SSH suite for the OpenBSD project. People quickly noticed that this powerful suite was secure, and most importantly, free. This led to the eventual porting across the various Unix flavors. In fact, many Unix projects today ship OpenSSH as part of their base system. We used the OpenSSH suite extensively throughout the writing of this book.

The official OpenSSH web site is http://www.openssh.com/. This site contains documentation and other useful information about the suite. You can download the suite of tools via FTP from http://www.openssh.com/portable.htm.

Top

A.6 Ethereal

Ethereal is a free network protocol analyzer for both Unix and Windows. It is a powerful analyzer that contains many useful features, including the ability to read network traces from virtually any other analyzer. It also boasts a rather impressive list of supported protocols that rivals most of the other available analyzers. Best of all, you can read its traces on most popular operating systems.

Ethereal includes a graphical interface and a text-based mode called Tethereal. Throughout this book, we used the CLI version to illustrate the behavior of various protocols. We highly recommend using this protocol analyzer.

The official Ethereal web site is http://www.ethereal.com/. This site contains documentation and other useful information about Ethereal. You can download the program via FTP from http://www.ethereal.com/download.html.

Top

Appendix B. IP Precedence, TOS, and DSCP Classifications

In Chapter 11, we discussed several important concepts related to traffic classification, queueing algorithms, and congestion handling systems. Unfortunately, some of these concepts are unfamiliar to many network engineers, so this appendix includes some more detail and background.

Every IP packet (including both IPv4 and IPv6) includes aTOS byte. This byte is broken up into fields that the network uses to help provide the appropriate QoS commitments. In the older TOS model defined in RFC 1349, the first three bits contain the IP Precedence value, and the next four bits contain the TOS value.

It is easy to get confused between the different uses of the term "TOS." Sometimes it refers to the entire byte and sometimes to just the four bits that describe forwarding behavior. To help reduce the confusion, we call the four-bit field the TOS field, and the entire byte the TOS byte.

Table B-1 shows the standard IP Precedence values. It is important to note that normal application traffic is not permitted to use IP Precedence values 6 or 7, which are strictly reserved for keepalive packets, routing protocols, and other important network traffic. The network must always give these packets higher priority than any application packets because no application will work if the network loses its topology information.

Table B-1. Standard IP Precedence values

IP Precedence	Decimal value	Bit pattern
Routine	0	000
Priority	1	001
Immediate	2	010
Flash	3	011
Flash Override	4	100
Critical	5	101
Internetwork control	6	110
Network control	7	111

Table B-2 shows the standard IP TOS values, as defined in RFC 1349. The idea was that an application could use these bits to request the appropriate forwarding behavior. Because the values are specified in different bits, the standard originally allowed applications to specify more than one option. This turned out to be unmanageable in practice, because it wasn't clear which bit should have precedence in cases where two bits were set, and each would select a different path. So the standard was changed in RFC 1349 to prevent combinations of TOS bits.

Table B-2. Standard IP TOS values

IP TOS	Decimal value	Bit pattern
Normal	0	0000
Minimum monetary cost	1	0001
Maximum reliability	2	0010
Maximum throughput	4	0100
Minimum delay	8	1000

Note that there is some disagreement in the literature about the last bit, which sometimes signifies "minimum monetary cost" and sometimes is not used at all. Some references state that the TOS byte has one unused bit, and others say that there are two unused bits. In any case, this entire scheme is now considered obsolete, and has been replaced by the DSCP model. However, many common applications (including Telnet and FTP) still set TOS field values by default. So it is important that the network be able to handle these settings gracefully.

In the new DSCP formalism, defined in RFC 2474, theTOS byte is divided into a 6-bit DSCP field followed by 2 unused bits. As we discuss in the next section, the DSCP formalism was designed to give good backward compatibility with the older formalism. In particular, the first three bits of the DSCP field map perfectly onto the older IP Precedence definitions.

The first three bits of the DSCP field identify the forwarding class. If the value in the first 3 bits is 4 or less, the packet uses Assured Forwarding (AF). If the value is 5, which corresponds to the highest allowed application IP Precedence value, then the packet uses Expedited Forwarding (EF). These names are slightly confusing because, in general, Assured Forwarding is merely expedient, while Expedited Forwarding is more likely to assure delivery.

Table B-3 shows the Assured Forwarding DSCP values. As we have already mentioned, the first 3 bits specify the forwarding class. A higher value in this sub-field results in a higher forwarding precedence through the network. The remaining 3 bits specify Drop Precedence. The higher the Drop Precedence, the more likely the packet will be dropped if it encounters congestion.

Table B-3. Assured Forwarding DSCP values

	Class 1		Class 2		Class 3		Class 4	
Drop Precedence	**Value**	**Name**	**Value**	**Name**	**Value**	**Name**	**Value**	**Name**
Lowest Drop Precedence	001010 (10)	AF11	010010 (18)	AF21	011010 (26)	AF31	100010 (34)	AF41
Medium Drop Precedence	001100 (12)	AF12	010100 (20)	AF22	011100 (28)	AF32	100100 (36)	AF42
Highest Drop Precedence	001110 (14)	AF13	010110 (22)	AF23	011110 (30)	AF33	100110 (38)	AF43

For Expedited Forwarding there is only one value. It has a binary value of `101110`, or `46` in decimal, and it is usually simply called EF. Note that this continues to follow the same pattern. The first 3 bits correspond to a decimal value of 5, which was the highest application IP Precedence value. You could think of the remaining three bits as specifying the highest Drop Precedence, but really this isn't meaningful because there is only one EF value. However, there is still significant room for defining additional EF types if it becomes necessary in the future.

The remaining two unused bits in the TOS byte have been the subject of some very interesting discussions lately. RFC 3168 suggests that they might be used for congestion notifications, similar to the Frame Relay FECN (Forward Explicit Congestion Notification) and BECN (Backward Explicit Congestion Notification) flags. This would seem to be a natural place to make this designation, since there is no congestion notification field anywhere else in the IPv4 or IPv6 headers. If packets carried this sort of information, routers could use adaptive processes to optimize forwarding behavior. If a link started to become congested, all upstream routers would automatically sense the problem and start to back off the rate that they were sending traffic before any application suffered from queue drops. This would be similar to the adaptive Frame Relay traffic shaping system that we discussed in Recipe 11.11. We look forward to seeing Cisco implement this feature in the future.

Top

B.1 Combining TOS and IP Precedence to Mimic DSCP

You can also get the equivalent of DSCP, even on older routers that support only TOS and Precedence, by combining the TOS and Precedence values. All Assured Forwarding DSCP Class 1 values are equivalent to an IP Precedence value of 1,*Priority*. All Class 2 values correspond to IP Precedence 2, *Immediate*; Class 3 values to IP Precedence 3,*Flash*; and Class 4 corresponds to an IP Precedence value of 4, *Flash Override*. The higher IP Precedence values are not used for Assured Forwarding.

You can then select the appropriate Drop Precedence group from the TOS values. However, you have to be careful, since there are 4 TOS bits. Combining this with the 3 bits from IP Precedence gives you 7 bits to work with, while DSCP only uses the first 6. For example, looking at the bit values that give AF11 in Table B-3, you can see that the last three bits are `010`. So the corresponding TOS value would be `0100`, which is 4 in decimal, or maximum throughput.

In Table B-3, you can see that a TOS value of 4, maximum throughput, always gives the lowest AF Drop Precedence. Selecting a TOS value of 8, minimum delay, gives medium Drop Precedence in all classes. And you can get the highest Assured Forwarding Drop Precedence value by setting a TOS value of 12, which doesn't have a standard name in the TOS terminology.

There is reasonably good interoperability between the AF DSCP variables and the combination of IP Precedence and TOS, which is good because it's impossible for a router to tell the difference in general. Only the three top priorities of IP Precedence are not represented, and that is simply because these DSCP values are used for guaranteed delivery services.

The biggest difference between the TOS and Assured Forwarding models is that, where the Assured Forwarding model is used to define a type of queueing, the TOS model is used to select a particular path. TOS was intended to work with a routing protocol, such as OSPF, to select the most appropriate path for a particular packet based on its TOS value. That is, when there are multiple paths available, a TOS-based OSPF (such as the version suggested in RFC 2676) would attempt to make a reasonable TOS assignment to each of them. If the router needed to forward a packet that was marked with a particular TOS value, it would attempt to find a route with the same TOS value. Note that Cisco never incorporated this type of functionality into its OSPF implementation, however. So, if TOS is going to have an effect on how packets are forwarded, you have to configure it manually by means of policy-based routing.

This was the historical intent for TOS, but in practice, most engineers found that it was easier to just use the TOS field to influence queueing behavior rather than path selection. So the IETF developed the more modern and useful DSCP formalism.

Assured Forwarding introduces the concept of Per-Hop Behavior (PHB). Each DSCP value has a corresponding well-defined PHB that the router uses not to select a path, but to define how it will

forward the packet. The router will forward a packet marked AF13 along the same network path as a packet marked AF41 if they both have the same destination address. However, it will be more likely to drop the AF13 packet if there is congestion, and it will forward the AF41 packet first if there are several packets in the queue.

From this it should be clear why it is easier to implement AF than TOS-based routing on a network.

Top

B.2 RSVP

Reservation Protocol (RSVP) is a signaling protocol that allows applications to request and reserve network resources, usually bandwidth. The core protocol is defined in RFC 2205. It is important to remember that RSVP is used only for requesting and managing network resources. RSVP does not carry user application data. Once the network has allocated the required resources, the application marks the packets for special treatment by setting the DSCP field to the EF value, `101110`.

The process starts when an end device sends an RSVP PATH request into the network. The destination address of this request is the far end device that it wants to communicate with. The request packet includes information about the application's source and destination addresses, protocol and port numbers, as well as its QoS requirements. It could specify a minimum required bandwidth, and perhaps also delay parameters. Each router along the path picks up this packet and figures out the best path to the destination.

Each router receiving an RSVP PATH request replaces the source address in the packet with its own, and forwards the packet to the next router along the path. So the QoS parameters are requested separately on each router-to-router hop. If a router is able to accommodate the request, it sends back an RSVP RESV message to the requester. For all but the first router on the path, the requester is the previous router. If a router receives one of these RESV packets, it knows that everything upstream from it is able to comply with the request. If it also has the resources to accommodate the requested QoS parameters, it sets aside the resources and sends an RESV packet to its downstream neighbor. It also sends an RSVP CONFIRM message upstream to acknowledge that the request will be honored. The routers periodically pass PATH, RESV, and CONFIRM packets to one another to ensure that the resources remain available.

If a router is not able to set aside the requested resources for whatever reason, it rejects the reservation. This may result in the entire path being rejected, but it can also just mean that the network will reserve the resources everywhere except on this one router-to-router link.

It would clearly be counterproductive if every device on the network could request as much bandwidth as they wanted, whenever they wanted. This would leave few network resources for routine applications. So usually when you configure a router for RSVP, you just set aside a relatively small fraction of the total bandwidth on a link for reservation. Further, you will often want to restrict which source addresses are permitted to make RSVP requests.

Because RSVP makes its reservation requests separately on each link, it can easily accommodate multicast flows. In this case, you have to be careful that the periodic updates happen quickly so that any new multicast group members won't have to wait long before they start to receive data. Please refer to

Chapter 23 for a more detailed discussion of multicast services.

RSVP is an extremely useful technique for reserving network resources for real-time applications such as Voice over IP (VoIP). However, because it forces the routers to keep detailed information on individual data flows, it doesn't scale well in large networks. RSVP is most useful at the edges of a large network, where you can reserve bandwidth entering the core. However, you probably don't want it running through the core of your network.

In large networks, it is common to use RSVP only at the edges of the network, with more conventional DSCP-based methods controlling QoS requirements in the core.

Top

B.3 Queueing Algorithms

You can implement several different queueing algorithms on Cisco routers. The most common type is Weighted Fair Queueing (WFQ), which is enabled by default on low-speed interfaces. There is also a class-based version of WFQ called Class-Based Weighted Fair Queueing (CBWFQ). These algorithms have the advantage of being fast, reliable, and easy to implement. However, in some cases, you might want to consider some of the other queueing systems available on Cisco routers.

Priority Queueing lets you specify absolute prioritization in your network so that more important packets always precede less important ones. This can be useful, but it is often dangerous in practice.

The other important queueing algorithm on Cisco routers is Custom Queueing, which allows you direct control over many of the queueing parameters.

B.3.1 Weighted Fair Queueing

A *flow* is loosely defined as the stream of packets associated with a single session of a single application. The common IP implementations of Fair Queueing (FQ) and WFQ assume that two packets are part of the same flow if they have the same source and destination IP addresses, the same source and destination TCP or UDP port numbers, and the same IP protocol field value. The algorithms combine these five values into a hash number, and sort the queued packets by this hash number.

The router then assigns sequence numbers to the queued packets. In the FQ algorithm, this process of sequencing the packets is optimized so that each flow gets a roughly equal share of the available bandwidth. As it receives each packet, the router assigns a sequence number based on the length of this packet and the total number of bytes associated with this same flow that are already in the queue.

This has a similar effect to a flow-based Round Robin (RR) queueing algorithm, in which all of the flows are assigned to different queues. These queues are then processed a certain number of bytes at a time until enough bytes have accumulated for a given queue to send a whole packet. Although this is a useful way of picturing the algorithm mentally, it is important to remember that the Cisco implementations of FQ and WFQ do not actually work this way. They keep all of the packets in a single queue. So, if there is a serious congestion problem, you will still get global tail drops.

This distinction is largely irrelevant for FQ, but for WFQ it's quite important. WFQ introduces another factor besides flow and packet size into the sequence numbers. The new factor is the *weight* (W), which is calculated from the IP Precedence (P) value. For IOS levels after 12.0(5)T, the formula is:

$$W = 32768/(P+1)$$

For all earlier IOS levels, the weight is lower by a factor of 4096:

$$W = 4096/(P+1)$$

Cisco increased the value to allow for finer control over weighting granularity.

The weight number for each packet is multiplied by the length of the packet when calculating sequence numbers. The result is that the router gives flows with higher IP Precedence values a larger share of the bandwidth than those with lower precedence. In fact, it is easy to calculate the relative scaling of the bandwidth shares of flows with different precedence values.

Table B-1Table B-1 shows, for example, that a flow with Flash Override Precedence will get five times the bandwidth of a packet with Routine Precedence. However, if all of the flows have the same precedence, WFQ behaves exactly the same as FQ.

Table B-4. Relative share of bandwidth in WFQ by IP Precedence

Precedence name	Value	Relative share of bandwidth
Routine	0	1
Priority	1	2
Immediate	2	3
Flash	3	4
Flash Override	4	5
Critical	5	6
Internetwork control	6	7
Network control	7	8

These algorithms tend to do three things. First, they prevent individual flows from interfering with one another. Second, they tend to reduce queueing latency for applications with smaller packets. Third, they ensure that all of the packets from a given flow are delivered in the same order that they were sent.

In practice, of course, a router has limited memory resources, so there is a limit to how many flows it can handle. If the number of flows is too large or the volume of traffic is too high, the router will start to have trouble with the computation. So these algorithms tend to be best on low-speed interfaces. WFQ is enabled by default on all interfaces with bandwidth of E1 (roughly 2Mbps) or less. The only exceptions are interfaces that use SDLC or LAPB link layer protocols, which require FIFO queuing.

Cisco provides several mechanisms to improve the bandwidth scaling of queueing algorithms. The first

is Distributed Weighted Fair Queueing (DWFQ), which is only available in routers that have Versatile Interface Processor (VIP) cards such as 7500 series routers, or the older 7000 series with RSP7000 processors. DWFQ is essentially the same as WFQ, except that the router is able to distribute the queueing calculations to the various VIP modules. But there is also another important difference: DWFQ uses a different sorting algorithm called Calendar Queueing, which uses much more memory, but operates much faster. This tradeoff means that you can use DWFQ on a VIP2-50 card containing Port Adapters Modules (PAM) with an aggregate line speed of up to OC-3. In fact, if the aggregate line speed is greater than a DS-3 (45Mbps), we don't recommend using DWFQ on anything slower than a VIP2-50. Cisco claims that DWFQ can operate at up to OC-3 speeds. However, if you need to support several interfaces that aggregate to OC-3 speeds on one VIP module, you may want to consider a different queueing strategy, particularly CBWFQ.

The next popular queueing strategy on Cisco routers, particularly for higher speed interfaces, is CBWFQ. CBWFQ is similar to WFQ, except that it doesn't group traffic by flows. Instead, it groups by traffic classes. A class is simply some logical grouping of traffic. It could be based on IP Precedence values, source addresses, input interface, or a variety of other locally useful rules that you can specify on the router.

The principal advantage to CBWFQ is that it allows you to expand the functionality of WFQ to higher speeds by eliminating the need to keep track of a large number of flows. But there is another important advantage to CBWFQ. The most common and sensible way to use CBWFQ is to assign the classes according to precedence or DSCP values. You can then manually adjust the weighting factors for the different classes. As you can see in Table B-1, the standard WFQ weighting factors give traffic with an IP Precedence value of 1 twice as much bandwidth as Precedence 0 traffic. However, Precedence 7 traffic gets just under 17% more bandwidth than Precedence 6 traffic. For many applications, these arbitrary weighting factors are not appropriate. So the ability to adjust these weighting factors can come in handy if you need to give your highest-priority traffic a larger share of the bandwidth.

B.3.2 Priority Queueing

Priority Queueing (PQ) is an older queueing algorithm that handles traffic with different precedence levels much more pragmatically. The Cisco implementation of Priority Queueing uses four distinct queues called *high priority*, *medium priority*, *normal priority*, and *low priority*. The PQ algorithm maintains an extremely strict concept of priority. If there are any packets in a higher priority queue, they must be sent first before any packets in the lower priority queues are sent.

Some types of critical real-time applications that absolutely cannot wait for low priority traffic work well with PQ. However, there is an obvious problem with this strategy: if the volume of traffic in the higher priority queues is greater than the link capacity, then no traffic from the lower priority queues will be forwarded. PQ *starves* low priority applications in these cases.

So a pure PQ implementation requires that you have an extremely good understanding of your traffic

patterns. The high priority traffic must represent a small fraction of the total, with the lowest priorities having the largest net volume. Further, you must have enough link capacity that the PQ algorithm is only used during peak bursts. If there is routine link congestion, PQ will give extremely poor overall performance.

However, Cisco has recently implemented a new hybrid queue type, called Low Latency Queueing (LLQ), which you can use with CBWFQ to give the best features of PQ while avoiding the queue starvation problem. The idea is simply to use CBWFQ for all of the traffic except for a small number subqueues that are reserved strictly for relatively low-volume real-time applications. The router services the real-time queues using a strict priority scheme and the others using CBWFQ. So, if there is a packet in one of the real-time queues, the router will transmit it before looking in one of the other queues. However, when there is nothing in the priority queues, the router will use normal CBWFQ for everything else.

LLQ also includes the stipulation that if the volume of high priority traffic exceeds a specified rate, the router will stop giving it absolute priority. The guarantees that LLQ will never starve the low priority queues.

This model is best suited to applications like voice or video where the real-time data comes in a fairly continuous but low bandwidth stream of small packets, as opposed to more bursty applications such as file transfers.

B.3.3 Custom Queueing

Custom Queueing (CQ) is one of Cisco's most popular queueing strategies. CQ was originally implemented to address the clear shortcomings of PQ. It lets you configure how many queues are to be used, what applications will use which queues, and how the queues will be serviced. Where PQ has only 4 queues, CQ allows you to use up to 16. And, perhaps most importantly, it includes a separate system queue so that user application data cannot starve critical network control traffic.

CQ is implemented as a round-robin queueing algorithm. The router takes a certain predetermined amount of data from each queue on each pass, which you can configure as a number of bytes. This allows you to specify approximately how much of the bandwidth each queue will receive. For example, if you have four queues, all set to the same number of bytes per pass, you can expect to send roughly equal amounts of data for all of these applications. Since the queues are used only when the network link is congested, this means that each of the four applications will receive roughly one quarter of the available bandwidth.

However, it is important to remember that the router will always take data one packet at a time. If you have a series of 1500-byte packets sitting in a particular queue and have configured the router to take 100 bytes from this queue on each pass, it will actually transmit 1 entire packet each time, and not 1 every 15 times. This is important because it can mean that your calculations of the relative amounts of

bandwidth allocated to each queue might be different from what the router actually sends.

This difference tends to disappear as you increase the number of bytes taken each time the queues are serviced. But you don't want to let the number get too large or you will cause unnecessary latency problems for your applications. For example, if the byte count for each of your four queues is 10,000 bytes, and all of the queues are full, the router will send 10,000 bytes from the first queue, then 10,000 bytes from the second queue, and so on. From the time it finishes servicing the first queue until the time that it returns to service it again, it will have sent 30,000 bytes. It takes roughly 160ms to send this much data through a T1 link, but the gap between the previous two packets in this queue was effectively zero. Variations in latency like this are called *jitter*, and they can cause serious problems for many real-time applications.

So, as with all of the other queueing algorithms we have discussed, CQ has some important advantages and disadvantages. Chapter 11 contains recipes that implement all of the queueing varieties we have discussed. You need to select the one that matches your network requirements best. None of them is perfect for all situations.

Top

B.4 Dropping Packets and Congestion Avoidance

Imagine a queue that holds packets as they enter a network bottleneck. These packets carry data for many different applications to many different destinations. If the amount of traffic arriving is less than the available bandwidth in the bottleneck, then the queue just holds the packets long enough to transmit them downstream. Queues become much more important if there is not enough bandwidth in the bottleneck to carry all of the incoming traffic.

If the excess is a short burst, the queue will attempt to smooth the flow rate, delivering the first packets as they are received and delaying the later ones briefly before transmitting them. However, if the burst is longer, or more like a continuous stream, the queue will have to stop accepting new packets while it deals with the backlog. The queue simply discards the overflowing inbound packets. This is called a *tail drop*.

Some applications and protocols deal with dropped packets more gracefully than others. For example, if an application doesn't have the ability to resend the lost information, then a dropped packet could be devastating. On the other hand, some real-time applications don't want their packets delayed. For these applications, it is better to drop the data than to delay it.

From the network's point of view, some protocols are better behaved than others. Applications that use TCP are able to adapt to dropping an occasional packet by backing off and sending data at a slower rate. However, many UDP-based protocols will simply send as many packets as they can stuff into the network. These applications will keep sending packets even if the network can't deliver them.

Even if all applications were TCP-based, however, there would still be some applications that take more than their fair share of network resources. If the only way to tell them to back off and send data more slowly is to wait until the queue fills up and starts to tail drop new packets, then it is quite likely that the wrong traffic flows will be instructed to slow down. However, an even worse problem called *global synchronization* can occur in an all-TCP network with a lot of tail drops.

Global synchronization happens when several different TCP flows all suffer packet drops simultaneously. Because the applications all use the same TCP mechanisms to control their flow rate, they will all back off in unison. TCP then starts to automatically increase the data rate until it suffers from more packet drops. Since all of the applications use the same algorithm for this process, they will all increase in unison until the tail drops start again. This whole wavelike oscillation of traffic rates will repeat as long as there is congestion.

Random Early Detection (RED) and its cousin, Weighted Random Early Detection (WRED), are two mechanisms to help avoid this type of problem, while at the same time keeping one flow from dominating. These algorithms assume that all of the traffic is TCP-based. This is important because

UDP applications get absolutely no benefit from RED or WRED.

RED and WRED try to prevent tail drops by preemptively dropping packets before the queue is full. If the link is not congested, then the queue is always more or less empty, so these algorithms don't do anything. However, when the queue depth reaches a minimum threshold, RED and WRED start to drop packets at random. The idea is to take advantage of the fact that TCP applications will back off their sending rate if they drop a packet. By randomly thinning out the queue before it becomes completely full, RED and WRED keep the TCP applications from overwhelming the queue and causing tail drops.

The packets to be dropped are selected at random. This has a couple of important advantages. First, the busiest flow is likely to be the one with the most packets in the queue, and therefore the most likely to suffer packet drops and be forced to back off. Second, by dropping packets at random, the algorithm effectively eliminates the global synchronization problems discussed earlier.

The probability of dropping a packet rises linearly with the queue depth, starting from a specified minimum threshold up to a maximum value. A simple example can help to explain how this works. Suppose the minimum threshold is set to 5 packets in the queue, and the maximum is set to 15 packets. If there are fewer than 5 packets in the queue, RED will not drop anything. When the queue depth reaches the maximum threshold, RED will drop one packet in 10. If there are 10 packets in the queue, then it is exactly halfway between the minimum and maximum thresholds and RED will drop half as many packets as it will at the maximum threshold: 1 packet in 20. Similarly, 7 packets in the queue represents 20% of the distance between the minimum and maximum thresholds, so the drop probability will be 20% of the maximum: 1 packet in 50.

If the queue fills up despite the random drops, then the router has no choice but to resort to tail drops, the same as if there were no sophisticated congestion avoidance. So RED and WRED have a particularly clever way of telling the difference between a momentary burst and longer-term heavy traffic volume, because they need to be much more aggressive with persistent congestion problems.

Instead of using a constant queue depth threshold value, these algorithms base the decision to drop packets on an exponential moving time averaged queue depth. If the queue fills because of a momentary burst of packets, RED will not start to drop packets immediately. However, if the queue continues to be busy for a longer period of time, the algorithm will be increasingly aggressive about dropping packets. This way the algorithm doesn't disrupt short bursts, but it will have a strong effect on applications that routinely overuse the network resources.

The WRED algorithm is similar to RED, except that it selectively prefers to drop packets that have lower IP Precedence values. Cisco routers achieve this by simply having a lower minimum threshold for lower precedence traffic. So, as the congestion increases, the router will tend to drop packets with lower precedence values. This tends to protect important traffic at the expense of less important applications. However, it is also important to bear in mind that this works best when the amount of high precedence traffic is relatively small.

If there is a lot of high priority traffic in the queue, it will not tend to benefit much from the efficiency improvements typically offered by WRED. In this case, you will likely see only a slight improvement over the characteristics of ordinary tail drops. This is yet another reason for being careful in your traffic categorization and not being too generous with the high precedence values.

Flow-based WRED is an interesting variant on WRED. In this case, the router makes an effort to separate out the individual flows in the router and penalize only those that are using more than their share of the bandwidth. The router does this by maintaining a separate drop probability for each flow based on their individual moving averages. The heaviest flows with the lowest precedence values tend to have the most dropped packets. However, it is important to note that the queue is congested by all the traffic, not just the heaviest flows. So the lighter flows will also have a finite drop probability in this situation. But, the fact that the heavy flow will have more packets in the queue, combined with the higher drop probability for these heavier flows, means that you should expect them to contribute most of the dropped packets.

Top

Colophon

Our look is the result of reader comments, our own experimentation, and feedback from distribution channels. Distinctive covers complement our distinctive approach to technical topics, breathing personality and life into potentially dry subjects.

The animal on the cover of Cisco Cookbook is a black jaguar (Panthera onca), sometimes called a black panther. While the color of black (melanastic) jaguars differs from that of the more common golden-yellow variety, they are of the same species. Jaguars of all types are native to the tropics, swamps, and grasslands of Central and South America (and rumored to still exist in parts of the southwestern U.S.), but the black jaguar is usually found only in dense forests. They are between 4 and 6 feet long and have a long tail that is usually about 30 inches long. Males can weigh up to 250 pounds, while females are considerably smaller and rarely grow to more than 150 pounds. Even though black jaguars often appear to be a solid black in artistic renditions and photography, their coats still have the dark rings containing even darker spots that are a distinguishing feature of all jaguars. Also notable are their eyes, which are a shiny reflective yellow.

Jaguars will eat almost any animal, including sloths, pigs, deer, monkeys, and cattle. Their hooked claw allow them to catch fish, frogs, turtles, and even small alligators. Even though they sit at the top of the rain forest food chain, humans are a large threat to jaguars of all colors-it's estimated that only 15,000 jaguars are left in the wild and the species is listed as near threatened. They are hunted for their coats (the black coat is greatly prized) and deforestation threatens their survival.

The black jaguar plays a large role in many South American religions, and is often considered a wise and divine animal who is associated with the worlds of magic and spirit. The Aztecs believed that the jaguar was the earthbound representative of their deity, and both the Mayans and Toltecs believed that their Sun God became a black jaguar at night in order to pass unseen through the underworld.

Philip Dangler was the production editor and copyeditor for Cisco Cookbook. Sarah Sherman, Derek Di Matteo, Jane Ellin, and Claire Cloutier provided quality control. Julie Hawks wrote the index. Jamie Peppard and Mary Agner provided production assistance.

Ellie Volckhausen designed the cover of this book, based on a series design by Edie Freedman. The cover image is a 19th-century engraving from the Dover Pictorial Archive. Emma Colby produced the cover layout with QuarkXPress 4.1 using Adobe's ITC Garamond font.

David Futato designed the interior layout. This book was converted by Andrew Savikas to FrameMaker 5.5.6 with a format conversion tool created by Erik Ray, Jason McIntosh, Neil Walls, and Mike Sierra that uses Perl and XML technologies. The text font is Linotype Birka; the heading font is Adobe Myriad Condensed; and the code font is LucasFont's TheSans Mono Condensed. The illustrations that appear in

the book were produced by Robert Romano and Jessamyn Read using Macromedia FreeHand 9 and Adobe Photoshop 6. The tip and warning icons were drawn by Christopher Bing. This colophon was written by Philip Dangler.

The online edition of this book was created by the Safari production group (John Chodacki, Becki Maisch, and Madeleine Newell) using a set of Frame-to-XML conversion and cleanup tools written and maintained by Erik Ray, Benn Salter, John Chodacki, and Jeff Liggett.

Top

[SYMBOL] [A] [B] [C] [D] [E] [F] [G] [H] [I] [K] [L] [M] [N] [O] [P] [Q] [R] [S] [T] [U] [V] [W] [X] [Z]

Top

[**SYMBOL**] [A] [B] [C] [D] [E] [F] [G] [H] [I] [K] [L] [M] [N] [O] [P] [Q] [R] [S] [T] [U] [V] [W] [X] [Z]

Top

Top

Top

Top

Top

Top

Top

Top

Top

Top

[SYMBOL] [A] [B] [C] [D] [E] [F] [G] [H] [I] [**K**] [L] [M] [N] [O] [P] [Q] [R] [S] [T] [U] [V] [W] [X] [Z]

Top

Top

Top

configuration files larger than

Top

Top

Top

Top

Top

Top

Top

Top

Top

Top

Top

zcat command

Top