ÿØÿà

Cisco Cookbook

By Ian J. Brown, Kevin Dooley

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

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.

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.

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.

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.

Previous Next

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.

Comments and Questions

Please address comments and questions about this book to the publisher:
O'Reilly & Associates, Inc.1005 Gravenstein Highway NorthSebastopol, 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

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.

Chapter 1. Router Configuration and File Management

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.

Previous Next

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
```

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
```

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
```

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
```

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:
```

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:
```

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
```

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
```

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
```

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:
```

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:
```

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
```

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
```

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
```

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
```

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
```

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
```

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 1-4 will 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
```

Chapter 2. Router Management

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.

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
```

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
```

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
```

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
```

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
```

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
```

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
```

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
```

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
```

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
```

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
```

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
```

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
```

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
```

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
```

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
```

Chapter 3. User Access and Privilege Levels

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 of Cisco 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.

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
```

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
```

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
```

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
```

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
```

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
```

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
```

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
```

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
```

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
```

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
```

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
```

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
```

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
```

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
```

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
```

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
```

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
```

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
```

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
```

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
```

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
```

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
```

Chapter 4. TACACS+

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.

Previous Next

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
```

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
```

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
```

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
```

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
```

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
```

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
```

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
```

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
```

Chapter 5. IP Routing

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.

Previous Next

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
```

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
```

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
```

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
```

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
```

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
```

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
```

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
```

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
```

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
```

Chapter 6. RIP

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.

Previous Next

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
```

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
```

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
```

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
```

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
```

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
```

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
```

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
```

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
```

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
```

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
```

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
```

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
```

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
```

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
```

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
```

Chapter 7. EIGRP

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.

Previous Next

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
```

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
```

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
```

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
```

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
```

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
```

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
```

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
```

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
```

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
```

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
```

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
```

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
```

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
```

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
```

Chapter 8. OSPF

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.

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
```

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
```

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
```

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
```

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
```

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
```

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
```

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
```

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
```

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
```

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
```

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
```

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
```

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
```

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
```

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
```

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
```

Chapter 9. BGP

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.

Previous Next

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
```

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
```

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
```

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
```

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
```

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
```

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
```

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
```

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
```

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
```

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
```

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
```

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
```

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
```

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
```

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
```

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
```

Chapter 10. Frame Relay

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.

Previous Next

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
```

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
```

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
```

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
```

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
```

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
```

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
```

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
```

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
```

Chapter 11. Queueing and Congestion

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.

Previous Next

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
```

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
```

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
```

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
```

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
```

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
```

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
```

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
```

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
```

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
```

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
```

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
```

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
```

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
```

Chapter 12. Tunnels and VPNs

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.

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
```

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
```

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
```

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
```

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
```

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
```

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
```

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
```

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
```

Chapter 13. Dial Backup

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:
$P(k,N) = N!\ Pk\ (1-P)(N-k) / (k!\ (N-k)!)$

The symbol "!" is a standard shorthand notation for the factorial function:
$N! = N \times (N - 1) \times (N-2) \times ... \times 2 \times 1$

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:
$P(2,100) \sim .5\%\ P(3,100) \sim .02\%\ P(4,100) \sim 0.00038\%\ P(10,100) \sim 1.7 \times 10\text{-}15\%$

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 $5 \times 10\text{-}10$ 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.

Previous Next

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
```

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
```

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
```

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
```

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
```

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
```

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
```

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
```

Chapter 14. NTP and Time

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.

Previous Next

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
```

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
```

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
```

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
```

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
```

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
```

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
```

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
```

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
```

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
```

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
```

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
```

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
```

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
```

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
```

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
```

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
```

Chapter 15. DLSw

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
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	1111 1111	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.

Previous Next

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
```

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
```

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
```

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
```

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
```

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
```

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
```

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

Previous Next

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
```

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
```

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
```

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
```

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
```

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
```

Chapter 16. Router Interfaces and Media

Introduction

Cisco supports a huge variety of different media types. There are over 50 different types of interface 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.

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
```

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
```

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
```

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
```

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
```

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
```

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
```

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
```

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
```

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
```

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
```

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
```

Chapter 17. Simple Network Management Protocol

Top

Introduction

Since its introduction in 1988, the Simple 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 (see Recipe 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.

◀ Previous Next ▶

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
```

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
```

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
```

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
```

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
```

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
```

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
```

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
```

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
```

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
```

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
```

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
```

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
```

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
```

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
```

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
```

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
```

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
```

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
```

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
```

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
```

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
```

Chapter 18. Logging

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
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
```

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
```

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
```

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
```

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
```

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
```

Recipe 18.6 Enabling Syslog on a Unix Server

18.6.1 Problem

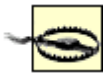
You want to configure a Unix server to accept syslog messages from routers.

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
```

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
```

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
```

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
```

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
```

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
```

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
```

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
```

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
```

Chapter 19. Access Lists

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
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).

Previous Next

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
```

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
```

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
```

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
```

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
```

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
```

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
```

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
```

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
```

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
```

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
```

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
```

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
```

Chapter 20. DHCP

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.

Previous Next

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
```

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
```

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
```

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
```

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
```

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
```

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
```

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
```

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
```

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
```

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
```

Chapter 21. NAT

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.

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
```

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
```

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
```

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
```

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
```

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
```

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
```

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
```

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 *
```

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
```

Chapter 22. Hot Standby Router Protocol

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
```

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
```

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
```

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
```

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
```

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
```

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
```

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
```

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
```

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
```

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
```

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
```

Chapter 23. IP Multicast

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]
```

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
```

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
```

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
```

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
```

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
```

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
```

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
```

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
```

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
```

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
```

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
```

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
```

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
```

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
```

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
```

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
```

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.

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).

Previous Next

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).

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.

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.

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.

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.

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 a TOS 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, the TOS 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.

Previous Next

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.

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.

Previous Next

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.

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.

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 claws 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.

Previous Next

Top

Previous Next

Top

Previous Next

Top
