CHAPTER 6

Using the Command-Line Interface

This chapter covers the following exam topics:

1.0 Network Fundamentals

1.6 Select the appropriate cabling type based on implementation requirements

NOTE This chapter primarily explains foundational skills required before you can explore the roughly 20 exam topics that use the verbs configure, verify, and troubleshoot.

To create an Ethernet LAN, a network engineer starts by planning. They consider the requirements, create a design, buy the switches, contract to install cables, and configure the switches to use the right features.

The CCENT and CCNA Routing and Switching exams focus on skills like understanding how LANs work, configuring different switch features, verifying that those features work correctly, and finding the root cause of the problem when a feature is not working correctly. The first skill you need to learn before doing all the configuration, verification, and troubleshooting tasks is to learn how to access and use the user interface of the switch, called the command-line interface (CLI).

This chapter begins that process by showing the basics of how to access the switch's CLI. These skills include how to access the CLI and how to issue verification commands to check on the status of the LAN. This chapter also includes the processes of how to configure the switch and how to save that configuration.

Note that this chapter focuses on processes that provide a foundation for most every exam topic that includes the verbs configure, verify, and troubleshoot. Chapter 7, "Analyzing Ethernet LAN Switching," Chapter 8, "Configuring Basic Switch Management," and Chapter 9, "Configuring Switch Interfaces," then examine particular commands you can use to verify and configure different switch features.

"Do I Know This Already?" Quiz

Take the quiz (either here, or use the PCPT software) if you want to use the score to help you decide how much time to spend on this chapter. The answers are at the bottom of the page following the quiz, and the explanations are in DVD Appendix C and in the PCPT software.

Table 6-1 "Do I Know This Already?" Foundation Topics Section-to-Question Mapping

Foundation Topics Section	Questions
Accessing the Cisco Catalyst Switch CLI	1–3
Configuring Cisco IOS Software	4–6

1. In what modes can you type the command **show mac address-table** and expect to get a response with MAC table entries? (Choose two answers.)
   a. User mode
   b. Enable mode
   c. Global configuration mode
   d. Interface configuration mode
2. In which of the following modes of the CLI could you type the command **reload** and expect the switch to reboot?
   a. User mode
   b. Enable mode
   c. Global configuration mode
   d. Interface configuration mode
3. Which of the following is a difference between Telnet and SSH as supported by a Cisco switch?
   a. SSH encrypts the passwords used at login, but not other traffic; Telnet encrypts nothing.
   b. SSH encrypts all data exchange, including login passwords; Telnet encrypts nothing.
   c. Telnet is used from Microsoft operating systems, and SSH is used from UNIX and Linux operating systems.
   d. Telnet encrypts only password exchanges; SSH encrypts all data exchanges.
4. What type of switch memory is used to store the configuration used by the switch when it is up and working?
   a. RAM
   b. ROM
   c. Flash
   d. NVRAM
   e. Bubble

5. What command copies the configuration from RAM into NVRAM?
   a. **copy running-config tftp**
   b. **copy tftp running-config**
   c. **copy running-config start-up-config**
   d. **copy start-up-config running-config**
   e. **copy startup-config running-config**
   f. **copy running-config startup-config**

6. A switch user is currently in console line configuration mode. Which of the following would place the user in enable mode? (Choose two answers.)
   a. Using the **exit** command once
   b. Using the **end** command once
   c. Pressing the Ctrl+Z key sequence once
   d. Using the **quit** command

Foundation Topics

Accessing the Cisco Catalyst Switch CLI

Cisco uses the concept of a command-line interface (CLI) with its router products and most of its Catalyst LAN switch products. The CLI is a text-based interface in which the user, typically a network engineer, enters a text command and presses Enter. Pressing Enter sends the command to the switch, which tells the device to do something. The switch does what the command says, and in some cases, the switch replies with some messages stating the results of the command.

Cisco Catalyst switches also support other methods to both monitor and configure a switch. For example, a switch can provide a web interface, so that an engineer can open a web browser to connect to a web server running in the switch. Switches also can be controlled and operated using network management software.

This book discusses only Cisco Catalyst enterprise-class switches, and in particular, how to use the Cisco CLI to monitor and control these switches. This first major section of the chapter first examines these Catalyst switches in more detail, and then explains how a network engineer can get access to the CLI to issue commands.

Cisco Catalyst Switches

Within the Cisco Catalyst brand of LAN switches, Cisco produces a wide variety of switch series or families. Each switch series includes several specific models of switches that have similar features, similar price-versus-performance trade-offs, and similar internal components.

For example, at the time this book was published, the Cisco 2960-X series of switches was a current switch model series. Cisco positions the 2960-X series (family) of switches as full-featured, low-cost wiring closet switches for enterprises. That means that you would expect to use 2960-X switches as access switches in a typical campus LAN design. Chapter 10, "Analyzing Ethernet LAN Designs," discusses campus LAN design and the roles of various switches.

Figure 6-1 shows a photo of 10 different models from the 2960-X switch model series from Cisco. Each switch series includes several models, with a mix of features. For example, some of the switches have 48 RJ-45 unshielded twisted-pair (UTP) 10/100/1000 ports, meaning that these ports can autonegotiate the use of 10BASE-T (10 Mbps), 100BASE-T (100 Mbps), or 1000BASE-T (1 Gbps) Ethernet.

Figure 6-1 *Cisco 2960-X Catalyst Switch Series*

Cisco refers to a switch's physical connectors as either *interfaces* or *ports*, with an interface type and interface number. The interface type, as used in commands on the switch, is either Ethernet, Fast Ethernet, Gigabit Ethernet, and so on for faster speeds. For Ethernet interfaces that support running at multiple speeds, the permanent name for the interface refers to the fastest supported speed. For example, a 10/100/1000 interface (that is, an interface that runs at 10 Mbps, 100 Mbps, or 1000 Mbps) would be called Gigabit Ethernet no matter what speed is currently in use.

To uniquely number each different interface, some Catalyst switches use a two-digit interface number (*x/y*), while others have a three-digit number (*x/y/z*). For instance, two 10/100/1000 ports on many older Cisco Catalyst switches would be called Gigabit Ethernet 0/0 and Gigabit Ethernet 0/1, while on the newer 2960-X series, two interfaces would be Gigabit Ethernet 1/0/1 and Gigabit Ethernet 1/0/2, for example.

Accessing the Cisco IOS CLI

Like any other piece of computer hardware, Cisco switches need some kind of operating system software. Cisco calls this OS the Internetwork Operating System (IOS).

Cisco IOS Software for Catalyst switches implements and controls logic and functions performed by a Cisco switch. Besides controlling the switch's performance and behavior, Cisco IOS also defines an interface for humans called the CLI. The Cisco IOS CLI allows the user to use a terminal emulation program, which accepts text entered by the user. When the user presses Enter, the terminal emulator sends that text to the switch. The switch processes the text as if it is a command, does what the command says, and sends text back to the terminal emulator.

The switch CLI can be accessed through three popular methods—the console, Telnet, and Secure Shell (SSH). Two of these methods (Telnet and SSH) use the IP network in which the switch resides to reach the switch. The console is a physical port built specifically to allow access to the CLI. Figure 6-2 depicts the options.

Answers to the "Do I Know This Already?" quiz:

1 A, B **2** B **3** B **4** A **5** F **6** B, C

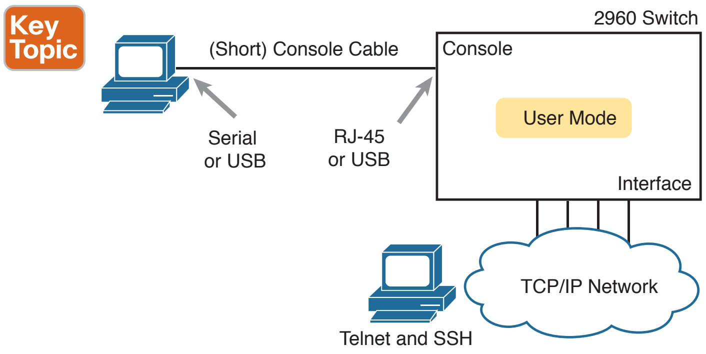


Figure 6-2 *CLI Access Options*

Console access requires both a physical connection between a PC (or other user device) and the switch's console port, as well as some software on the PC. Telnet and SSH require software on the user's device, but they rely on the existing TCP/IP network to transmit data. The next few pages detail how to connect the console and set up the software for each method to access the CLI.

Cabling the Console Connection

The physical console connection, both old and new, uses three main components: the physical console port on the switch, a physical serial port on the PC, and a cable that works with the console and serial ports. However, the physical cabling details have changed slowly over time, mainly because of advances and changes with serial interfaces on PC hardware. For this next topic, the text looks at three cases: newer connectors on both the PC and the switch, older connectors on both, and a third case with the newer (USB) connector on the PC but with an older connector on the switch.

More modern PC and switch hardware use a familiar standard USB cable for the console connection. Cisco has been including USB ports as console ports in newer routers and switches as well. All you have to do is look at the switch to make sure you have the correct style of USB cable end to match the USB console port. In the simplest form, you can use any USB port on the PC, with a USB cable, connected to the USB console port on the switch or router, as shown on the far right side of Figure 6-3.

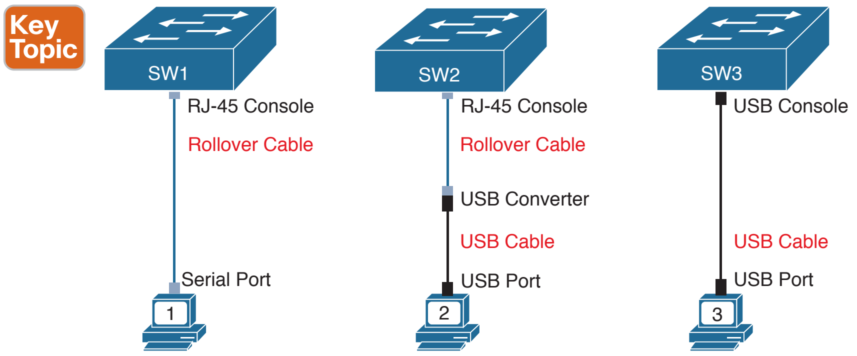


Figure 6-3 *Console Connection to a Switch*

Older console connections use a PC serial port that pre-dates USB, a UTP cable, and an RJ-45 console port on the switch, as shown on the left side of Figure 6-3. The PC serial port typically has a D-shell connector (roughly rectangular) with nine pins (often called a DB-9). The console port looks like any Ethernet RJ-45 port (but is typically colored in blue and with the word "console" beside it on the switch).

The cabling for this older-style console connection can be simple or require some effort, depending on what cable you use. You can use the purpose-built console cable that ships with new Cisco switches and routers and not think about the details. However, you can make your own cable with a standard serial cable (with a connector that matches the PC), a standard RJ-45 to DB-9 converter plug, and a UTP cable. However, the UTP cable does not use the same pinouts as Ethernet; instead, the cable uses rollover cable pinouts rather than any of the standard Ethernet cabling pinouts. The rollover pinout uses eight wires, rolling the wire at pin 1 to pin 8, pin 2 to pin 7, pin 3 to pin 6, and so on.

As it turns out, USB ports became common on PCs before Cisco began commonly using USB for its console ports. So, you also have to be ready to use a PC that has only a USB port and not an old serial port, but a router or switch that has the older RJ-45 console port (and no USB console port). The center of Figure 6-3 shows that case. To connect such a PC to a router or switch console, you need a USB converter that converts from the older console cable to a USB connector, and a rollover UTP cable, as shown in the middle of Figure 6-3.

NOTE When using the USB options, you typically also need to install a software driver so that your PC's OS knows that the device on the other end of the USB connection is the console of a Cisco device. Also, you can easily find photos of these cables and components online, with searches like "cisco console cable," "cisco usb console cable," or "console cable converter."

The newer 2960-X series, for instance, supports both the older RJ-45 console port and a USB console port. Figure 6-4 points to the two console ports; you would use only one or the other. Note that the USB console port uses a mini-B port rather than the more commonly seen rectangular standard USB port.

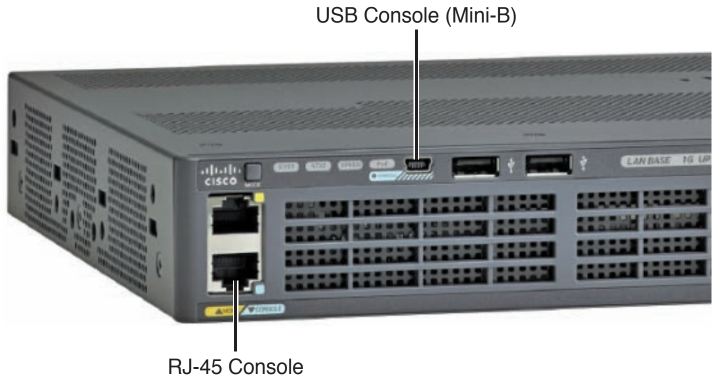


Figure 6-4 *A Part of a 2960-X Switch with Console Ports Shown*

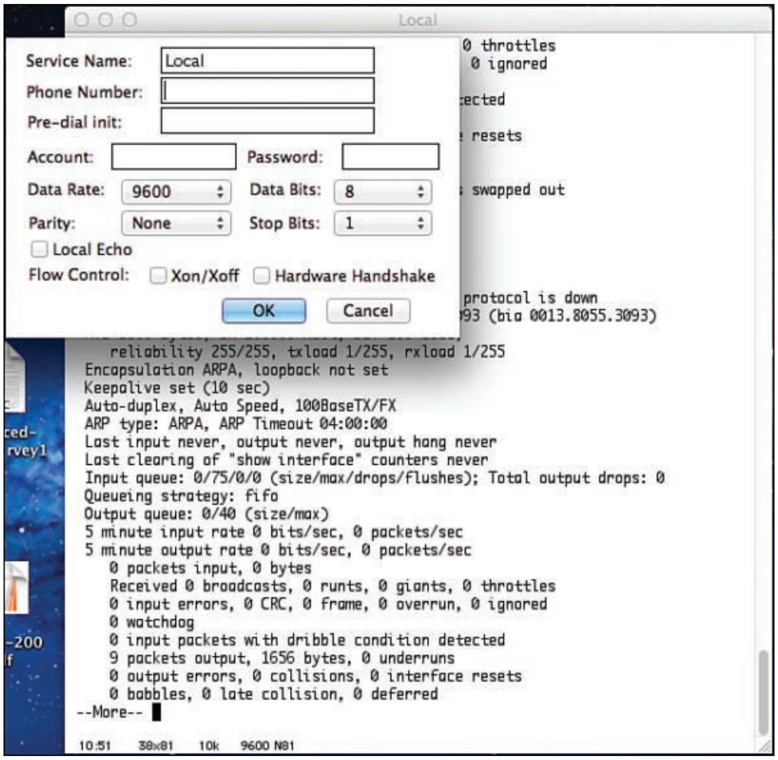
After the PC is physically connected to the console port, a terminal emulator software package must be installed and configured on the PC. The terminal emulator software treats all data as text. It accepts the text typed by the user and sends it over the console connection to the switch. Similarly, any bits coming into the PC over the console connection are displayed as text for the user to read.

The emulator must be configured to use the PC's serial port to match the settings on the switch's console port settings. The default console port settings on a switch are as follows. Note that the last three parameters are referred to collectively as 8N1:

Key Topic

- 9600 bits/second
- No hardware flow control
- 8-bit ASCII
- No parity bits
- 1 stop bit

Figure 6-5 shows one such terminal emulator. The image shows the window created by the emulator software in the background, with some output of a **show** command. The foreground, in the upper left, shows a settings window that lists the default console settings as listed just before this paragraph.



Service Name: Local
Phone Number:
Pre-dial init:
Account: Password:
Data Rate: 9600 Data Bits: 8
Parity: None Stop Bits: 1
Local Echo
Flow Control: Xon/Xoff Hardware Handshake

```
0 throttles
 0 ignored
ected
 resets
 swapped out
protocol is down
93 (bia 0013.8055.3093)
   reliability 255/255, txload 1/255, rxload 1/255
  Encapsulation ARPA, loopback not set
  Keepalive set (10 sec)
  Auto-duplex, Auto Speed, 100BaseTX/FX
  ARP type: ARPA, ARP Timeout 04:00:00
  Last input never, output never, output hang never
  Last clearing of "show interface" counters never
  Input queue: 0/75/0/0 (size/max/drops/flushes); Total output drops: 0
  Queueing strategy: fifo
  Output queue: 0/40 (size/max)
  5 minute input rate 0 bits/sec, 0 packets/sec
  5 minute output rate 0 bits/sec, 0 packets/sec
     0 packets input, 0 bytes
     Received 0 broadcasts, 0 runts, 0 giants, 0 throttles
     0 input errors, 0 CRC, 0 frame, 0 overrun, 0 ignored
     0 watchdog
     0 input packets with dribble condition detected
     9 packets output, 1656 bytes, 0 underruns
     0 output errors, 0 collisions, 0 interface resets
     0 babbles, 0 late collision, 0 deferred
 --More--
```

10:51 38x81 10k 9600 N81

Figure 6-5 *Terminal Settings for Console Access*

Accessing the CLI with Telnet and SSH

For many years, terminal emulator applications have supported far more than the ability to communicate over a serial port to a local device (like a switch's console). Terminal emulators support a variety of TCP/IP applications as well, including Telnet and SSH. Telnet and SSH both allow the user to connect to another device's CLI, but instead of connecting through a console cable to the console port, the traffic flows over the same IP network that the networking devices are helping to create.

Telnet uses the concept of a Telnet client (the terminal application) and a Telnet server (the switch in this case). A *Telnet client*, the device that sits in front of the user, accepts keyboard input and sends those commands to the *Telnet server*. The Telnet server accepts the text, interprets the text as a command, and replies back. Telnet is a TCP-based application layer protocol that uses well-known port 23.

Cisco Catalyst switches enable a Telnet server by default, but switches need a few more configuration settings before you can successfully use Telnet to connect to a switch. Chapter 8 covers switch configuration to support Telnet and SSH in detail.

Using Telnet in a lab today makes sense, but Telnet poses a significant security risk in production networks. Telnet sends all data (including any username and password for login to the switch) as clear-text data. SSH gives us a much better option.

Think of SSH as the much more secure Telnet cousin. Outwardly, you still open a terminal emulator, connect to the switch's IP address, and see the switch CLI, no matter whether you use Telnet or SSH. The differences exist behind the scenes: SSH encrypts the contents of all messages, including the passwords, avoiding the possibility of someone capturing packets in the network and stealing the password to network devices. Like Telnet, SSH uses TCP, just using well-known port 22 instead of Telnet's 23.

User and Enable (Privileged) Modes

All three CLI access methods covered so far (console, Telnet, and SSH) place the user in an area of the CLI called *user EXEC mode*. User EXEC mode, sometimes also called *user mode*, allows the user to look around but not break anything. The "EXEC mode" part of the name refers to the fact that in this mode, when you enter a command, the switch executes the command and then displays messages that describe the command's results.

NOTE If you have not used the CLI before, you might want to experiment with the CLI from the Sim Lite product, or view the video about CLI basics. You can find these resources on the DVD and on the companion website, as mentioned in the introduction.

Cisco IOS supports a more powerful EXEC mode called *enable* mode (also known as *privileged* mode or *privileged EXEC* mode). Enable mode gets its name from the **enable** command, which moves the user from user mode to enable mode, as shown in Figure 6-6. The other name for this mode, *privileged mode*, refers to the fact that powerful (or privileged) commands can be executed there. For example, you can use the **reload** command, which tells the switch to reinitialize or reboot Cisco IOS, only from enable mode.

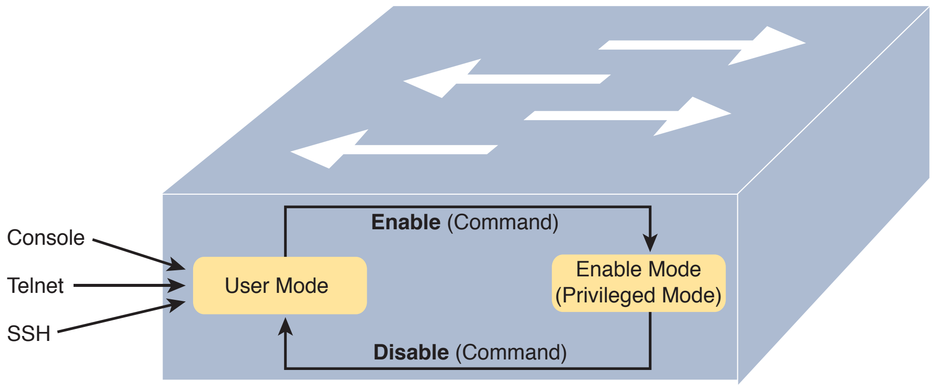


Figure 6-6 *User and Privileged Modes*

NOTE If the command prompt lists the hostname followed by a >, the user is in user mode; if it is the hostname followed by the #, the user is in enable mode.

Example 6-1 demonstrates the differences between user and enable modes. The example shows the output that you could see in a terminal emulator window, for instance, when connecting from the console. In this case, the user sits at the user mode prompt ("Certskills1>") and tries the **reload** command. The **reload** command tells the switch to reinitialize or reboot Cisco IOS, so IOS allows this powerful command to be used only from enable mode. IOS rejects the **reload** command when used in user mode. Then the user moves to enable mode—also called privileged mode—(using the **enable** EXEC command). At that point, IOS accepts the **reload** command now that the user is in enable mode.

Example 6-1 *Example of Privileged Mode Commands Being Rejected in User Mode*

```
Press RETURN to get started.

User Access Verification

Password:
Certskills1>
Certskills1> reload
Translating "reload"
% Unknown command or computer name, or unable to find computer address
Certskills1> enable
Password:
Certskills1#
Certskills1# reload

Proceed with reload? [confirm] y
00:08:42: %SYS-5-RELOAD: Reload requested by console. Reload Reason: Reload Command.
```

NOTE The commands that can be used in either user (EXEC) mode or enable (EXEC) mode are called EXEC commands.

This example is the first instance of this book showing you the output from the CLI, so it is worth noting a few conventions. The bold text represents what the user typed, and the non-bold text is what the switch sent back to the terminal emulator. Also, the typed passwords do not show up on the screen for security purposes. Finally, note that this switch has been preconfigured with a hostname of Certskills1, so the command prompt on the left shows that hostname on each line.

Password Security for CLI Access from the Console

A Cisco switch, with default settings, remains relatively secure when locked inside a wiring closet, because by default, a switch allows console access only. By default, the console requires no password at all, and no password to reach enable mode for users that happened to connect from the console. The reason is that if you have access to the physical console port of the switch, you already have pretty much complete control over the switch. You could literally get out your screwdriver and walk off with it, or you could unplug the power, or follow well-published procedures to go through password recovery to break into the CLI and then configure anything you want to configure.

However, many people go ahead and set up simple password protection for console users. Simple passwords can be configured at two points in the login process from the console: when the user connects from the console, and when any user moves to enable mode (using the **enable** EXEC command). You may have noticed that back in Example 6-1, the user saw a password prompt at both points.

Example 6-2 shows the additional configuration commands that were configured prior to collecting the output in Example 6-1. The output holds an excerpt from the EXEC command **show running-config**, which lists the current configuration in the switch.

Example 6-2 *Nondefault Basic Configuration*

```
Certskills1# show running-config
! Output has been formatted to show only the parts relevant to this discussion
hostname Certskills1
!
enable secret love
!
line console 0
 login
 password faith
! The rest of the output has been omitted
Certskills1#
```

Working from top to bottom, note that the first configuration command listed by the **show running-config** command sets the switch's hostname to Certskills1. You might have noticed that the command prompts in Example 6-1 all began with Certskills1, and that's why the command prompt begins with the hostname of the switch.

Next, note that the lines with a **!** in them are comment lines, both in the text of this book and in the real switch CLI.

The **enable secret love** configuration command defines the password that all users must use to reach enable mode. So, no matter whether a user connects from the console, Telnet, or SSH, they would use password love when prompted for a password after typing the **enable** EXEC command.

Finally, the last three lines configure the console password. The first line (**line console 0**) is the command that identifies the console, basically meaning "these next commands apply to the console only." The **login** command tells IOS to perform simple password checking (at the console). Remember, by default, the switch does not ask for a password for console users. Finally, the **password faith** command defines the password the console user must type when prompted.

This example just scratches the surface of the kinds of security configuration you might choose to configure on a switch, but it does give you enough detail to configure switches in your lab and get started (which is the reason I put these details in this first chapter of Part II). Note that Chapter 8 shows the configuration steps to add support for Telnet and SSH (including password security), and Chapter 34, "Device Security Features," shows additional security configuration as well.

CLI Help Features

If you printed the Cisco IOS Command Reference documents, you would end up with a stack of paper several feet tall. No one should expect to memorize all the commands—and no one does. You can use several very easy, convenient tools to help remember commands and save time typing. As you progress through your Cisco certifications, the exams will cover progressively more commands. However, you should know the methods of getting command help.

Table 6-2 summarizes command-recall help options available at the CLI. Note that, in the first column, *command* represents any command. Likewise, *parm* represents a command's parameter. For example, the third row lists *command* **?**, which means that commands such as **show ?** and **copy ?** would list help for the **show** and **copy** commands, respectively.

Table 6-2 Cisco IOS Software Command Help

What You Enter	What Help You Get
?	Help for all commands available in this mode.
command **?**	With a space between the command and the ?, the switch lists text to describe all the first parameter options for the command.
com?	A list of commands that start with **com**.
command parm?	Lists all parameters beginning with the **parameter typed so far.** (Notice that there is no space between *parm* and the **?**.)
command parm<Tab>	Pressing the Tab key causes IOS to spell out the rest of the word, assuming that you have typed enough of the word so there is only one option that begins with that string of characters.
command parm1 **?**	If a space is inserted before the question mark, the CLI lists all the next parameters and gives a brief explanation of each.

When you enter the **?**, the Cisco IOS CLI reacts immediately; that is, you don't need to press the Enter key or any other keys. The device running Cisco IOS also redisplays what you entered before the **?** to save you some keystrokes. If you press Enter immediately after the **?**, Cisco IOS tries to execute the command with only the parameters you have entered so far.

The information supplied by using help depends on the CLI mode. For example, when **?** is entered in user mode, the commands allowed in user mode are displayed, but commands available only in enable mode (not in user mode) are not displayed. Also, help is available in configuration mode, which is the mode used to configure the switch. In fact, configuration mode has many different subconfiguration modes, as explained in the section "Configuration Submodes and Contexts," later in this chapter. So, you can get help for the commands available in each configuration submode as well. (Note that this might be a good time to use the free NetSim Lite product on the DVD—open any lab, use the question mark, and try some commands.)

Cisco IOS stores the commands that you enter in a history buffer, storing ten commands by default. The CLI allows you to move backward and forward in the historical list of commands and then edit the command before reissuing it. These key sequences can help you use the CLI more quickly on the exams. Table 6-3 lists the commands used to manipulate previously entered commands.

Table 6-3 Key Sequences for Command Edit and Recall

Keyboard Command	What Happens
Up arrow or Ctrl+P	This displays the most recently used command. If you press it again, the next most recent command appears, until the history buffer is exhausted. (The *P* stands for previous.)
Down arrow or Ctrl+N	If you have gone too far back into the history buffer, these keys take you forward to the more recently entered commands. (The *N* stands for next.)
Left arrow or Ctrl+B	This moves the cursor backward in the currently displayed command without deleting characters. (The *B* stands for back.)
Right arrow or Ctrl+F	This moves the cursor forward in the currently displayed command without deleting characters. (The *F* stands for forward.)
Backspace	This moves the cursor backward in the currently displayed command, deleting characters.

The debug and show Commands

By far, the single most popular Cisco IOS command is the **show** command. The **show** command has a large variety of options, and with those options, you can find the status of almost every feature of Cisco IOS. Essentially, the **show** command lists the currently known facts about the switch's operational status. The only work the switch does in reaction to **show** commands is to find the current status and list the information in messages sent to the user.

For example, consider the output from the **show mac address-table dynamic** command listed in Example 6-3. This **show** command, issued from user mode, lists the table the switch uses to make forwarding decisions. A switch's MAC address table basically lists the data a switch uses to do its primary job.

Example 6-3 *Nondefault Basic Configuration*

```
Certskills1> show mac address-table dynamic
          Mac Address Table
-------------------------------------------

Vlan    Mac Address       Type        Ports
----    -----------       --------    -----
  31    0200.1111.1111    DYNAMIC     Gi0/1
  31    0200.3333.3333    DYNAMIC     Fa0/3
  31    1833.9d7b.0e9a    DYNAMIC     Gi0/1
  10    1833.9d7b.0e9a    DYNAMIC     Gi0/1
  10    30f7.0d29.8561    DYNAMIC     Gi0/1
   1    1833.9d7b.0e9a    DYNAMIC     Gi0/1
  12    1833.9d7b.0e9a    DYNAMIC     Gi0/1
Total Mac Addresses for this criterion: 7
Certskills1>
```

The **debug** command also tells the user details about the operation of the switch. However, while the **show** command lists status information at one instant of time—more like a photograph—the **debug** command acts more like a live video camera feed. Once you issue a **debug** command, IOS remembers, issuing messages that any switch user can choose to see. The console sees these messages by default. Most of the commands used throughout this book to verify operation of switches and routers are **show** commands.

Configuring Cisco IOS Software

You will want to configure every switch in an Enterprise network, even though the switches will forward traffic even with default configuration. This section covers the basic configuration processes, including the concept of a configuration file and the locations in which the configuration files can be stored. Although this section focuses on the configuration process, and not on the configuration commands themselves, you should know all the commands covered in this chapter for the exams, in addition to the configuration processes.

Configuration mode is another mode for the Cisco CLI, similar to user mode and privileged mode. User mode lets you issue non-disruptive commands and displays some information. Privileged mode supports a superset of commands compared to user mode, including commands that might disrupt switch operations. However, none of the commands in user or privileged mode changes the switch's configuration. Configuration mode accepts *configuration commands*—commands that tell the switch the details of what to do and how to do it. Figure 6-7 illustrates the relationships among configuration mode, user EXEC mode, and privileged EXEC mode.

Key Topic

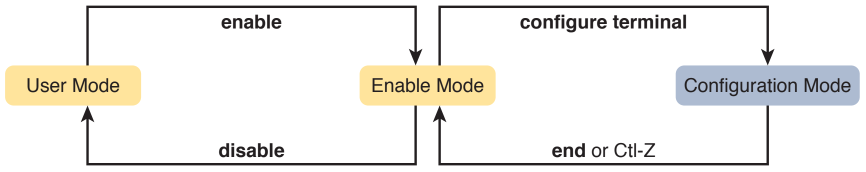


Figure 6-7 *CLI Configuration Mode Versus EXEC Modes*

Commands entered in configuration mode update the active configuration file. *These changes to the configuration occur immediately each time you press the Enter key at the end of a command.* Be careful when you enter a configuration command!

Configuration Submodes and Contexts

Configuration mode itself contains a multitude of commands. To help organize the configuration, IOS groups some kinds of configuration commands together. To do that, when using configuration mode, you move from the initial mode—global configuration mode—into subcommand modes. *Context-setting commands* move you from one configuration subcommand mode, or context, to another. These context-setting commands tell the switch the topic about which you will enter the next few configuration commands. More importantly, the context tells the switch the topic you care about right now, so when you use the **?** to get help, the switch gives you help about that topic only.

NOTE *Context-setting* is not a Cisco term. It is just a description used here to help make sense of configuration mode.

The best way to learn about configuration submodes is to use them, but first, take a look at these upcoming examples. For instance, the **interface** command is one of the most commonly used context-setting configuration commands. For example, the CLI user could enter interface configuration mode by entering the **interface FastEthernet 0/1** configuration command. Asking for help in interface configuration mode displays only commands that are useful when configuring Ethernet interfaces. Commands used in this context are called *subcommands*—or, in this specific case, *interface subcommands*. When you begin practicing with the CLI with real equipment, the navigation between modes can become natural. For now, consider Example 6-4, which shows the following:

- Movement from enable mode to global configuration mode by using the **configure terminal** EXEC command
- Using a **hostname Fred** global configuration command to configure the switch's name
- Movement from global configuration mode to console line configuration mode (using the **line console 0** command)
- Setting the console's simple password to **hope** (using the **password hope** line subcommand)
- Movement from console configuration mode to interface configuration mode (using the **interface** *type number* command)
- Setting the speed to 100 Mbps for interface Fa0/1 (using the **speed 100** interface subcommand)
- Movement from interface configuration mode back to global configuration mode (using the **exit** command)

Example 6-4 *Navigating Between Different Configuration Modes*

```
Switch# configure terminal
Switch(config)# hostname Fred
Fred(config)# line console 0
Fred(config-line)# password hope
Fred(config-line)# interface FastEthernet 0/1
Fred(config-if)# speed 100
Fred(config-if)# exit
Fred(config)#
```

The text inside parentheses in the command prompt identifies the configuration mode. For example, the first command prompt after you enter configuration mode lists (config), meaning global configuration mode. After the **line console 0** command, the text expands to (config-line), meaning line configuration mode. Each time the command prompt changes within config mode, you have moved to another configuration mode.

Table 6-4 shows the most common command prompts in configuration mode, the names of those modes, and the context-setting commands used to reach those modes.

Key Topic

Table 6-4 Common Switch Configuration Modes

Prompt	Name of Mode	Context-Setting Command(s) to Reach This Mode
hostname(config)#	Global	None—first mode after **configure terminal**
hostname(config-line)#	Line	**line console 0** **line vty 0 15**
hostname(config-if)#	Interface	**interface** *type number*
hostname(vlan)#	VLAN	**vlan** *number*

You should practice until you become comfortable moving between the different configuration modes, back to enable mode, and then back into the configuration modes. However, you can learn these skills just doing labs about the topics in later chapters of the book. For now, Figure 6-8 shows most of the navigation between global configuration mode and the four configuration submodes listed in Table 6-4.

Key Topic

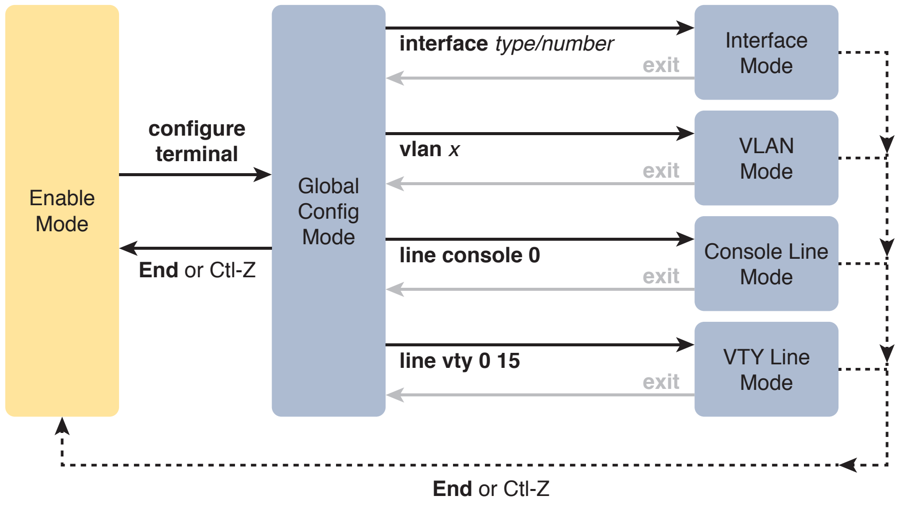


Figure 6-8 *Navigation In and Out of Switch Configuration Modes*

NOTE You can also move directly from one configuration submode to another, without first using the **exit** command to move back to global configuration mode. Just use the commands listed in bold in the center of the figure.

You really should stop and try navigating around these configuration modes. If you have not yet decided on a lab strategy, spin the DVD in the back of the book, and install the Pearson Sim Lite software. It includes the simulator and a couple of lab exercises. Start any lab, ignore the instructions, and just get into configuration mode and move around between the configuration modes shown in Figure 6-8.

No set rules exist for what commands are global commands or subcommands. Generally, however, when multiple instances of a parameter can be set in a single switch, the command used to set the parameter is likely a configuration subcommand. Items that are set once for the entire switch are likely global commands. For example, the **hostname** command is a global command because there is only one hostname per switch. Conversely, the **speed** command is an interface subcommand that applies to each switch interface that can run at different speeds, so it is a subcommand, applying to the particular interface under which it is configured.

Storing Switch Configuration Files

When you configure a switch, it needs to use the configuration. It also needs to be able to retain the configuration in case the switch loses power. Cisco switches contain random-access memory (RAM) to store data while Cisco IOS is using it, but RAM loses its contents when the switch loses power or is reloaded. To store information that must be retained when the switch loses power or is reloaded, Cisco switches use several types of more permanent memory, none of which has any moving parts. By avoiding components with moving parts (such as traditional disk drives), switches can maintain better uptime and availability.

The following list details the four main types of memory found in Cisco switches, as well as the most common use of each type:

- **RAM:** Sometimes called DRAM, for dynamic random-access memory, RAM is used by the switch just as it is used by any other computer: for working storage. The running (active) configuration file is stored here.
- **Flash memory:** Either a chip inside the switch or a removable memory card, flash memory stores fully functional Cisco IOS images and is the default location where the switch gets its Cisco IOS at boot time. Flash memory also can be used to store any other files, including backup copies of configuration files.
- **ROM:** Read-only memory (ROM) stores a bootstrap (or boothelper) program that is loaded when the switch first powers on. This bootstrap program then finds the full Cisco IOS image and manages the process of loading Cisco IOS into RAM, at which point Cisco IOS takes over operation of the switch.
- **NVRAM:** Nonvolatile RAM (NVRAM) stores the initial or startup configuration file that is used when the switch is first powered on and when the switch is reloaded.

Figure 6-9 summarizes this same information in a briefer and more convenient form for memorization and study.

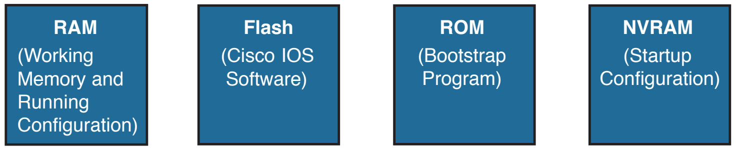


Figure 6-9 *Cisco Switch Memory Types*

Cisco IOS stores the collection of configuration commands in a *configuration file*. In fact, switches use multiple configuration files—one file for the initial configuration used when powering on, and another configuration file for the active, currently used running configuration as stored in RAM. Table 6-5 lists the names of these two files, their purpose, and their storage location.

Key Topic

Table 6-5 Names and Purposes of the Two Main Cisco IOS Configuration Files

Configuration Filename	Purpose	Where It Is Stored
startup-config	Stores the initial configuration used anytime the switch reloads Cisco IOS.	NVRAM
running-config	Stores the currently used configuration commands. This file changes dynamically when someone enters commands in configuration mode.	RAM

Essentially, when you use configuration mode, you change only the running-config file. This means that the configuration example earlier in this chapter (Example 6-4) updates only the running-config file. However, if the switch lost power right after that example, all that configuration would be lost. If you want to keep that configuration, you have to copy the running-config file into NVRAM, overwriting the old startup-config file.

Example 6-5 demonstrates that commands used in configuration mode change only the running configuration in RAM. The example shows the following concepts and steps:

Step 1. The example begins with both the running and startup-config having the same hostname, per the **hostname hannah** command.

Step 2. The hostname is changed in configuration mode using the **hostname jessie** command.

Step 3. The **show running-config** and **show startup-config** commands show the fact that the hostnames are now different, with the **hostname jessie** command found only in the running-config.

Example 6-5 *How Configuration Mode Commands Change the Running-Config File, Not the Startup-Config File*

```
! Step 1 next (two commands)
!
hannah# show running-config
! (lines omitted)
hostname hannah
! (rest of lines omitted)
```

```
hannah# show startup-config
! (lines omitted)
hostname hannah
! (rest of lines omitted)
! Step 2 next. Notice that the command prompt changes immediately after
! the hostname command.

hannah# configure terminal
hannah(config)# hostname jessie
jessie(config)# exit
! Step 3 next (two commands)
!
jessie# show running-config
! (lines omitted) - just showing the part with the hostname command
hostname jessie
!
jessie# show startup-config
! (lines omitted) - just showing the part with the hostname command
hostname hannah
```

Copying and Erasing Configuration Files

The configuration process updates the running-config file, which is lost if the router loses power or is reloaded. Clearly, IOS needs to provide us a way to copy the running configuration so that it will not be lost, so it will be used the next time the switch reloads or powers on. For instance, Example 6-5 ended with a different running configuration (with the **hostname jessie** command) versus the startup configuration.

In short, the EXEC command **copy running-config startup-config** backs up the running-config to the startup-config file. This command overwrites the current startup-config file with what is currently in the running-configuration file.

In addition, in lab, you may want to just get rid of all existing configuration and start over with a clean configuration. To do that, you can erase the startup-config file using three different commands:

```
write erase
erase startup-config
erase nvram:
```

Once the startup-config file is erased, you can reload or power off/on the switch, and it will boot with the now-empty startup configuration.

Note that Cisco IOS does not have a command that erases the contents of the running-config file. To clear out the running-config file, simply erase the startup-config file, and then **reload** the switch, and the running-config will be empty at the end of the process.

NOTE Cisco uses the term *reload* to refer to what most PC operating systems call rebooting or restarting. In each case, it is a re-initialization of the software. The **reload** EXEC command causes a switch to reload.

Chapter Review

One key to doing well on the exams is to perform repetitive spaced review sessions. Review this chapter's material using either the tools in the book, DVD, or interactive tools for the same material found on the book's companion website. Refer to the "Your Study Plan" element section titled "Step 2: Build Your Study Habits Around the Chapter" for more details. Table 6-6 outlines the key review elements and where you can find them. To better track your study progress, record when you completed these activities in the second column.

Table 6-6 Chapter Review Tracking

Review Element	Review Date(s)	Resource Used
Review key topics		Book, DVD/website
Review key terms		Book, DVD/website
Repeat DIKTA questions		Book, PCPT
Review memory tables		Book, DVD/website
Review command tables		Book

Review All the Key Topics

Key Topic

Table 6-7 Key Topics for Chapter 6

Key Topic Element	Description	Page Number
Figure 6-2	Three methods to access a switch CLI	130
Figure 6-3	Cabling options for a console connection	130
List	A Cisco switch's default console port settings	132
Figure 6-7	Navigation between user, enable, and global config modes	138
Table 6-4	A list of configuration mode prompts, the name of the configuration mode, and the command used to reach each mode	140
Figure 6-8	Configuration mode context-setting commands	140
Table 6-5	The names and purposes of the two configuration files in a switch or router	142

Key Terms You Should Know

command-line interface (CLI), Telnet, Secure Shell (SSH), enable mode, user mode, configuration mode, startup-config file, running-config file

Command References

Tables 6-8 and 6-9 list configuration and verification commands used in this chapter, respectively. As an easy review exercise, cover the left column in a table, read the right column, and try to recall the command without looking. Then repeat the exercise, covering the right column, and try to recall what the command does.

Table 6-8 Chapter 6 Configuration Commands

Command	Mode and Purpose
line console 0	Global command that changes the context to console configuration mode.
login	Line (console and vty) configuration mode. Tells IOS to prompt for a password (no username).
password *pass-value*	Line (console and vty) configuration mode. Sets the password required on that line for login if the **login** command (with no other parameters) is also configured.
interface *type port-number*	Global command that changes the context to interface mode—for example, **interface FastEthernet 0/1**.
hostname *name*	Global command that sets this switch's hostname, which is also used as the first part of the switch's command prompt.
exit	Moves back to the next higher mode in configuration mode.
end	Exits configuration mode and goes back to enable mode from any of the configuration submodes.
Ctrl+Z	This is not a command, but rather a two-key combination (pressing the Ctrl key and the letter Z) that together do the same thing as the **end** command.

Table 6-9 Chapter 6 EXEC Command Reference

Command	Purpose
no debug all **undebug all**	Enable mode EXEC command to disable all currently enabled debugs.
reload	Enable mode EXEC command that reboots the switch or router.
copy running-config startup-config	Enable mode EXEC command that saves the active config, replacing the startup-config file used when the switch initializes.
copy startup-config running-config	Enable mode EXEC command that merges the startup-config file with the currently active config file in RAM.
show running-config	Lists the contents of the running-config file.
write erase **erase startup-config** **erase nvram:**	These enable mode EXEC commands erase the startup-config file.
quit	EXEC command that disconnects the user from the CLI session.
show startup-config	Lists the contents of the startup-config (initial config) file.
enable	Moves the user from user mode to enable (privileged) mode and prompts for a password if one is configured.
disable	Moves the user from enable mode to user mode.
configure terminal	Enable mode command that moves the user into configuration mode.

5. A Cisco Catalyst switch has 24 10/100 ports, numbered 0/1 through 0/24. Ten PCs connect to the ten lowest numbered port, with those PCs working and sending data over the network. The other ports are not connected to any device. Which of the following answers lists facts displayed by the **show interfaces status** command?

- **a.** Port Ethernet 0/1 is in a connected state.
- **b.** Port Fast Ethernet 0/11 is in a connected state.
- **c.** Port Fast Ethernet 0/5 is in a connected state.
- **d.** Port Ethernet 0/15 is in a notconnected state.

6. Consider the following output from a Cisco Catalyst switch:

```
SW1# show mac address-table dynamic
          Mac Address Table
-------------------------------------------

Vlan     Mac Address         Type         Ports
----     -----------         --------     -----
   1     02AA.AAAA.AAAA     DYNAMIC      Gi0/1
   1     02BB.BBBB.BBBB     DYNAMIC      Gi0/2
   1     02CC.CCCC.CCCC     DYNAMIC      Gi0/3
Total Mac Addresses for this criterion: 3
```

Which of the following answers are true about this switch?

- **a.** The output proves that port Gi0/2 connects directly to a device that uses address 02BB.BBBB.BBBB.
- **b.** The switch has learned three MAC addresses since the switch powered on.
- **c.** The three listed MAC addresses were learned based on the destination MAC address of frames forwarded by the switch.
- **d.** 02CC.CCCC.CCCC was learned from the source MAC address of a frame that entered port Gi0/3.

Foundation Topics

LAN Switching Concepts

A modern Ethernet LAN connects user devices as well as servers into some switches, with the switches then connecting to each other, sometimes in a design like Figure 7-1. Part of the LAN, called a campus LAN, supports the end user population as shown on the left of the figure. End user devices connect to LAN switches, which in turn connect to other switches so that a path exists to the rest of the network. The campus LAN switches sit in wiring closets close to the end users. On the right, the servers used to provide information to the users also connects to the LAN. Those servers and switches often sit in a closed room called a *data center*, with connections to the campus LAN to support traffic to/from the users.

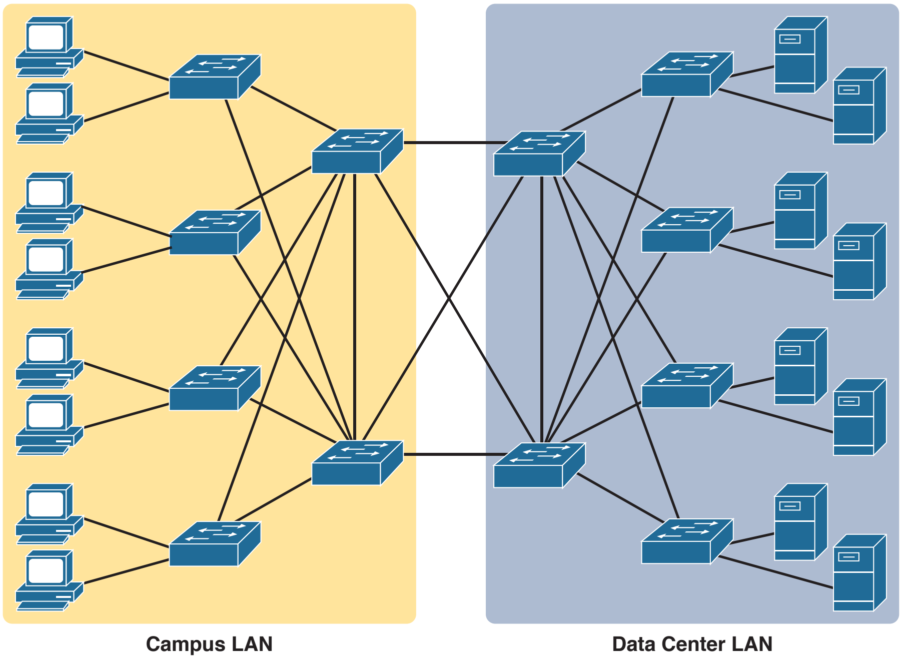


Figure 7-1 *Campus LAN and Data Center LAN, Conceptual Drawing*

To forward traffic from a user device to a server and back, each switch performs the same kind of logic, independently from each other. The first half of this chapter examines the logic: how a switch chooses to forward an Ethernet frame, when the switch chooses to not forward the frame, and so on.

Overview of Switching Logic

Ultimately, the role of a LAN switch is to forward Ethernet frames. LANs exist as a set of user devices, servers, and other devices that connect to switches, with the switches connected to each other. The LAN switch has one primary job: to forward frames to the correct destination (MAC) address. And to achieve that goal, switches use logic—logic based on the source and destination MAC address in each frame's Ethernet header.

Answers to the "Do I Know This Already?" quiz:

1 A **2** C **3** A **4** B **5** C **6** D

LAN switches receive Ethernet frames and then make a switching decision: either forward the frame out some other ports or ignore the frame. To accomplish this primary mission, switches perform three actions:

Key Topic

1. Deciding when to forward a frame or when to filter (not forward) a frame, based on the destination MAC address
2. Preparing to forward frames by learning MAC addresses by examining the source MAC address of each frame received by the switch
3. Preparing to forward only one copy of the frame to the destination by creating a (Layer 2) loop-free environment with other switches by using Spanning Tree Protocol (STP)

The first action is the switch's primary job, whereas the other two items are overhead functions.

NOTE Throughout this book's discussion of LAN switches, the terms *switch port* and *switch interface* are synonymous.

Although Chapter 2's section titled "Ethernet Data-Link Protocols" already discussed the frame format, this discussion of Ethernet switching is pretty important, so reviewing the Ethernet frame at this point might be helpful. Figure 7-2 shows one popular format for an Ethernet frame. Basically, a switch would take the frame shown in the figure, make a decision of where to forward the frame, and send the frame out that other interface.

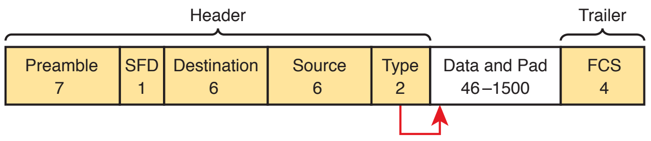


Figure 7-2 *IEEE 802.3 Ethernet Frame (One Variation)*

Most of the upcoming discussions and figures about Ethernet switching focuses on the use of the destination and source MAC address fields in the header. All Ethernet frames have both a destination and source MAC address. Both are 6-bytes long (represented as 12 hex digits in the book), and are a key part of the switching logic discussed in this section. Refer back to Chapter 2's discussion of the header in detail for more info on the rest of the Ethernet frame.

NOTE The companion DVD and website include a video that explains the basics of Ethernet switching.

Now on to the details of how Ethernet switching works!

Forwarding Known Unicast Frames

To decide whether to forward a frame, a switch uses a dynamically built table that lists MAC addresses and outgoing interfaces. Switches compare the frame's destination MAC address to this table to decide whether the switch should forward a frame or simply ignore it. For example, consider the simple network shown in Figure 7-3, with Fred sending a frame to Barney.

Key Topic

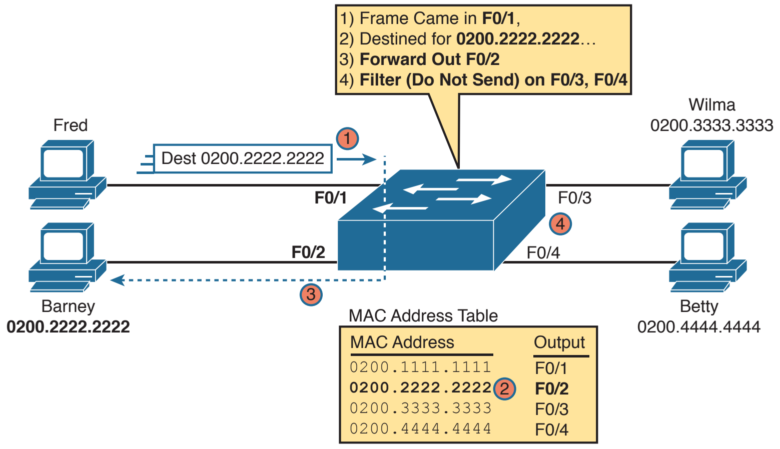


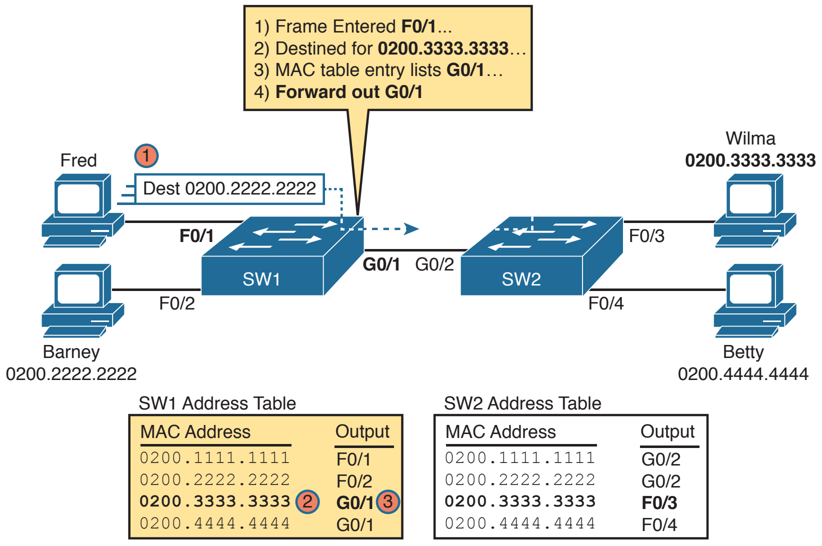
Figure 7-3 *Sample Switch Forwarding and Filtering Decision*

In this figure, Fred sends a frame with destination address 0200.2222.2222 (Barney's MAC address). The switch compares the destination MAC address (0200.2222.2222) to the MAC address table, matching the bold table entry. That matched table entry tells the switch to forward the frame out port F0/2, and only port F0/2.

NOTE A switch's MAC address table is also called the *switching table*, or *bridging table*, or even the *Content-Addressable Memory (CAM) table*, in reference to the type of physical memory used to store the table.

A switch's MAC address table lists the location of each MAC relative to that one switch. In LANs with multiple switches, each switch makes an independent forwarding decision based on its own MAC address table. Together, they forward the frame so that it eventually arrives at the destination.

For example, Figure 7-4 shows the first switching decision in a case in which Fred sends a frame to Wilma, with destination MAC 0200.3333.3333. The topology has changed versus the previous figure, this time with two switches, and Fred and Wilma connected to two different switches. Figure 7-3 shows the first switch's logic, in reaction to Fred sending the original frame. Basically, the switch receives the frame in port F0/1, finds the destination MAC (0200.3333.3333) in the MAC address table, sees the outgoing port of G0/1, so SW1 forwards the frame out its G0/1 port.



SW1 Address Table

MAC Address	Output
0200.1111.1111	F0/1
0200.2222.2222	F0/2
0200.3333.3333	**G0/1**
0200.4444.4444	G0/1

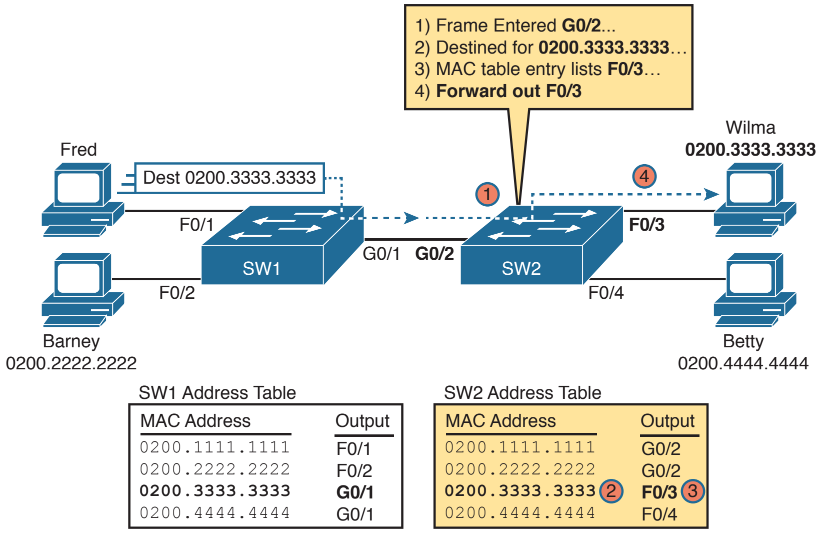
SW2 Address Table

MAC Address	Output
0200.1111.1111	G0/2
0200.2222.2222	G0/2
0200.3333.3333	**F0/3**
0200.4444.4444	F0/4

Figure 7-4 *Forwarding Decision with Two Switches: First Switch*

That same frame next arrives at switch SW2, entering SW2's G0/2 interface. As shown in Figure 7-5, SW2 uses the same logic steps, but using SW2's table. The MAC table lists the forwarding instructions for that switch only. In this case, switch SW2 forwards the frame out its F0/3 port, based on SW2's MAC address table.

Key Topic



SW1 Address Table

MAC Address	Output
0200.1111.1111	F0/1
0200.2222.2222	F0/2
0200.3333.3333	**G0/1**
0200.4444.4444	G0/1

SW2 Address Table

MAC Address	Output
0200.1111.1111	G0/2
0200.2222.2222	G0/2
0200.3333.3333	**F0/3**
0200.4444.4444	F0/4

Figure 7-5 *Forwarding Decision with Two Switches: Second Switch*

NOTE The forwarding choice by a switch was formerly called a *forward-versus-filter* decision, because the switch also chooses to not forward (to filter) frames, not sending the frame out some ports.

The examples so far use switches that happen to have a MAC table with all the MAC addresses listed. As a result, the destination MAC address in the frame is known to the switch. The frames are called known unicast frames, or simply known unicasts, because the destination address is a unicast address, and the destination is known. As shown in these examples, switches forward known unicast frames out one port: the port as listed in the MAC table entry for that MAC address.

Learning MAC Addresses

Thankfully, the networking staff does not have to type in all those MAC table entries. Instead, the switches do their second main function: to learn the MAC addresses and interfaces to put into its address table. With a complete MAC address table, the switch can make accurate forwarding and filtering decisions as just discussed.

Switches build the address table by listening to incoming frames and examining the *source MAC address* in the frame. If a frame enters the switch and the source MAC address is not in the MAC address table, the switch creates an entry in the table. That table entry lists the interface from which the frame arrived. Switch learning logic is that simple.

Figure 7-6 depicts the same single-switch topology network as Figure 7-3, but before the switch has built any address table entries. The figure shows the first two frames sent in this network—first a frame from Fred, addressed to Barney, and then Barney's response, addressed to Fred.

Key Topic

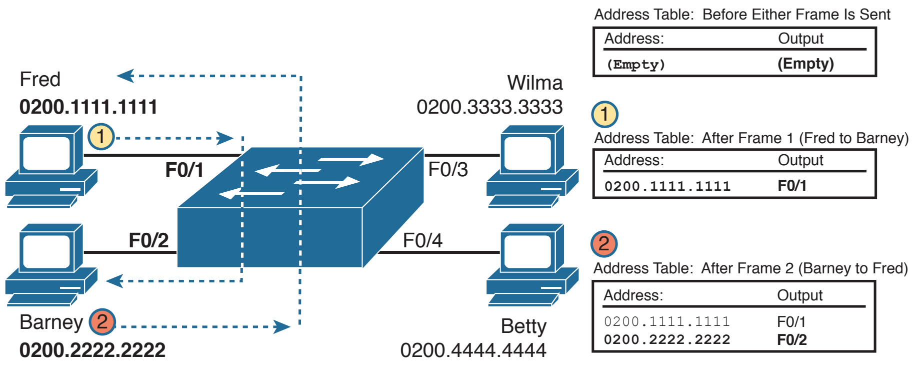


Figure 7-6 *Switch Learning: Empty Table and Adding Two Entries*

(Figure 7-6 depicts the MAC learning process only, and ignores the forwarding process and therefore ignores the destination MAC addresses.)

Focus on the learning process and how the MAC table grows at each step as shown on the right side of the figure. The switch begins with an empty MAC table, as shown in the upper right part of the figure. Then Fred sends his first frame (labeled "1") to Barney, so the switch adds an entry for 0200.1111.1111, Fred's MAC address, associated with interface F0/1. Why F0/1? The frame sent by Fred entered the switch's F0/1 port. SW1's logic runs something like this: "The source is MAC 0200.1111.1111, the frame entered F0/1, so from my perspective, 0200.1111.1111 must be reachable out my port F0/1."

Continuing the example, when Barney replies in Step 2, the switch adds a second entry, this one for 0200.2222.2222, Barney's MAC address, along with interface F0/2. Why F0/2? The frame Barney sent entered the switch's F0/2 interface. Learning always occurs by looking at the source MAC address in the frame, and adds the incoming interface as the associated port.

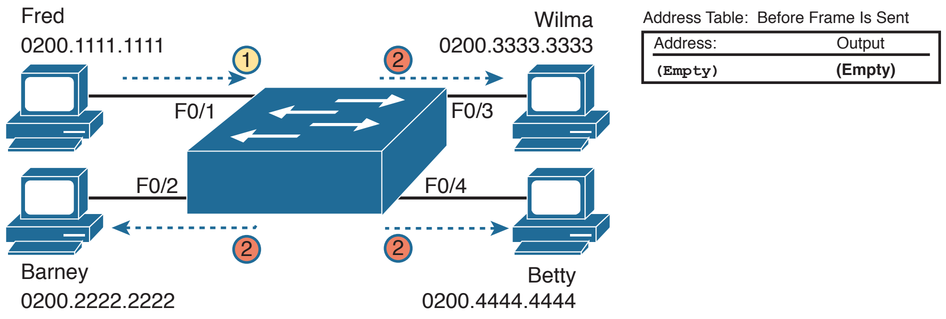
Flooding Unknown Unicast and Broadcast Frames

Now again turn your attention to the forwarding process, using the topology in Figure 7-5. What do you suppose the switch does with Fred's first frame, the one that occurred when there were no entries in the MAC address table? As it turns out, when there is no matching entry in the table, switches forward the frame out all interfaces (except the incoming interface) using a process called *flooding*. And the frame whose destination address is unknown to the switch is called an *unknown unicast frame*, or simply an *unknown unicast*.

Switches flood unknown unicast frames. Flooding means that the switch forwards copies of the frame out all ports, except the port on which the frame was received. The idea is simple: if you do not know where to send it, send it everywhere, to deliver the frame. And, by the way, that device will likely then send a reply—and then the switch can learn that device's MAC address, and forward future frames out one port as a known unicast frame.

Switches also flood LAN broadcast frames (frames destined to the Ethernet broadcast address of FFFF.FFFF.FFFF), because this process helps deliver a copy of the frame to all devices in the LAN.

For example, Figure 7-7 shows the same first frame sent by Fred, when the switch's MAC table is empty. At step 1, Fred sends the frame. At step 2, the switch sends a copy of the frame out all three of the other interfaces.



Address:	Output
(Empty)	(Empty)

Figure 7-7 *Switch Flooding: Unknown Unicast Arrives, Floods out Other Ports*

Avoiding Loops Using Spanning Tree Protocol

The third primary feature of LAN switches is loop prevention, as implemented by Spanning Tree Protocol (STP). Without STP, any flooded frames would loop for an indefinite period of time in Ethernet networks with physically redundant links. To prevent looping frames, STP blocks some ports from forwarding frames so that only one active path exists between any pair of LAN segments.

The result of STP is good: Frames do not loop infinitely, which makes the LAN usable. However, STP has negative features as well, including the fact that it takes some work to balance traffic across the redundant alternate links.

A simple example makes the need for STP more obvious. Remember, switches flood unknown unicast frames and broadcast frames. Figure 7-8 shows an unknown unicast frame, sent by Larry to Bob, which loops forever because the network has redundancy but no STP. Note that the figure shows one direction of the looping frame only, just to reduce clutter, but a copy of the frame would also loop the other direction as well.

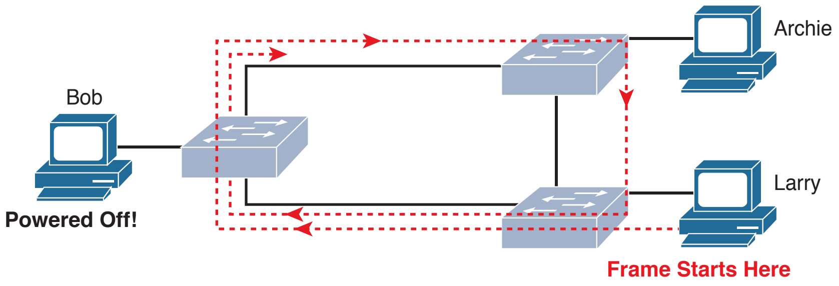


Figure 7-8 *Network with Redundant Links but Without STP: The Frame Loops Forever*

The flooding of this frame would cause the frame to rotate around the three switches, because none of the switches list Bob's MAC address in their address tables, each switch floods the frame. And while the flooding process is a good mechanism for forwarding unknown unicasts and broadcasts, the continual flooding of traffic frames as in the figure can completely congest the LAN to the point of making it unusable.

A topology like Figure 7-8, with redundant links, is good, but we need to prevent the bad effect of those looping frames. To avoid Layer 2 loops, all switches need to use STP. STP causes each interface on a switch to settle into either a blocking state or a forwarding state. *Blocking* means that the interface cannot forward or receive data frames, while *forwarding* means that the interface can send and receive data frames. If a correct subset of the interfaces is blocked, only a single currently active logical path exists between each pair of LANs.

NOTE STP behaves identically for a transparent bridge and a switch. Therefore, the terms *bridge*, *switch*, and *bridging device* all are used interchangeably when discussing STP.

The *Cisco CCNA Routing and Switching ICND2 200-105 Official Cert Guide* book covers the details of how STP prevents loops.

LAN Switching Summary

Switches use Layer 2 logic, examining the Ethernet data-link header to choose how to process frames. In particular, switches make decisions to forward and filter frames, learn MAC addresses, and use STP to avoid loops, as follows:

Key Topic

Step 1. Switches forward frames based on the destination MAC address:

A. If the destination MAC address is a broadcast, multicast, or unknown destination unicast (a unicast not listed in the MAC table), the switch floods the frame.

B. If the destination MAC address is a known unicast address (a unicast address found in the MAC table):

i. If the outgoing interface listed in the MAC address table is different from the interface in which the frame was received, the switch forwards the frame out the outgoing interface.

ii. If the outgoing interface is the same as the interface in which the frame was received, the switch filters the frame, meaning that the switch simply ignores the frame and does not forward it.

Step 2. Switches use the following logic to learn MAC address table entries:

A. For each received frame, examine the source MAC address and note the interface from which the frame was received.

B. If it is not already in the table, add the MAC address and interface it was learned on.

Step 3. Switches use STP to prevent loops by causing some interfaces to block, meaning that they do not send or receive frames.

Verifying and Analyzing Ethernet Switching

A Cisco Catalyst switch comes from the factory ready to switch frames. All you have to do is connect the power cable, plug in the Ethernet cables, and the switch starts switching incoming frames. Connect multiple switches together, and they are ready to forward frames between the switches as well. And the big reason behind this default behavior has to do with the default settings on the switches.

Cisco Catalyst switches come ready to get busy switching frames because of settings like these:

- The interfaces are enabled by default, ready to start working once a cable is connected.
- All interfaces are assigned to VLAN 1.
- 10/100 and 10/100/1000 interfaces use autonegotiation by default.
- The MAC learning, forwarding, flooding logic all works by default.
- STP is enabled by default.

This second section of the chapter examines how switches will work with these default settings, showing how to verify the Ethernet learning and forwarding process.

Demonstrating MAC Learning

To see a switches MAC address table, use the **show mac address-table** command. With no additional parameters, this command lists all known MAC addresses in the MAC table, including some overhead static MAC addresses that you can ignore. To see all the dynamically learned MAC addresses only, instead use the **show mac address-table dynamic** command.

The examples in this chapter use almost no configuration, as if you just unboxed the switch when you first purchased it. For the examples, the switches have no configuration other than the **hostname** command to set a meaningful hostname. Note that to do this in lab, all I did was

- Use the **erase startup-config** EXEC command to erase the startup-config file
- Use the **delete vlan.dat** EXEC command to delete the VLAN configuration details
- Use the **reload** EXEC command to reload the switch (thereby using the empty startup-config, with no VLAN information configured)
- Configure the **hostname SW1** command to set the switch hostname

Once done, the switch starts forwarding and learning MAC address, as demonstrated in Example 7-1.

Key Topic

Example 7-1 show mac address-table dynamic *for Figure 7-7*

```
SW1# show mac address-table dynamic
          Mac Address Table
-------------------------------------------


Vlan    Mac Address       Type        Ports
----    -----------       --------    -----
   1    0200.1111.1111    DYNAMIC     Fa0/1
   1    0200.2222.2222    DYNAMIC     Fa0/2
   1    0200.3333.3333    DYNAMIC     Fa0/3
   1    0200.4444.4444    DYNAMIC     Fa0/4
Total Mac Addresses for this criterion: 4
SW1#
```

First, focus on two columns of the table: the Mac Address and Ports columns of the table. The values should look familiar: they match the earlier single-switch example, as repeated here as Figure 7-9. Note the four MAC addresses listed, along with their matching ports, as shown in the figure.

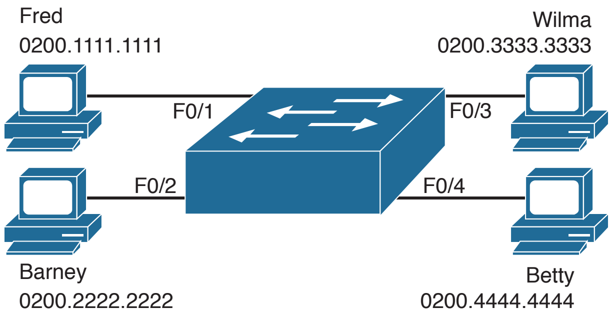


Figure 7-9 *Single Switch Topology Used in Verification Section*

Next, look at the Type field in the header. The column tells us whether the MAC address was learned by the switch as described earlier in this chapter. You can also statically predefine MAC table entries using a couple of different features, including port security, and those would appear as Static in the Type column.

Finally, the VLAN column of the output gives us a chance to briefly discuss how VLANs impact switching logic. LAN switches forward Ethernet frames inside a VLAN. What that means is if a frame enters via a port in VLAN 1, then the switch will forward or flood that frame out other ports in VLAN 1 only, and not out any ports that happen to be assigned to another VLAN. Chapter 11, "Implementing Ethernet Virtual LANs," looks at all the details of how switches forward frames when using VLANs.

Two key commands give some information about the status of SSH on the switch. First, the **show ip ssh** command lists status information about the SSH server itself. The **show ssh** command then lists information about each SSH client currently connected into the switch. Example 8-6 shows samples of each, with user Wendell currently connected to the switch.

Example 8-6 *Displaying SSH Status*

```
SW1# show ip ssh
SSH Enabled - version 2.0
Authentication timeout: 120 secs; Authentication retries: 3

SW1# show ssh
Connection Version Mode  Encryption    Hmac              State             Username
0          2.0     IN    aes128-cbc    hmac-sha1         Session started   wendell
0          2.0     OUT   aes128-cbc    hmac-sha1         Session started   wendell
%No SSHv1 server connections running.
```

Enabling IPv4 for Remote Access

To allow Telnet or SSH access to the switch, and to allow other IP-based management protocols (for example, Simple Network Management Protocol, or SNMP) to function as intended, the switch needs an IP address, as well as a few other related settings. The IP address has nothing to do with how switches forward Ethernet frames; it simply exists to support overhead management traffic.

This next topic begins by explaining the IPv4 settings needed on a switch, followed by the configuration. Note that although switches can be configured with IPv6 addresses with commands similar to those shown in this chapter, this chapter focuses solely on IPv4. All references to IP in this chapter imply IPv4.

Host and Switch IP Settings

A switch needs the same kind of IP settings as a PC with a single Ethernet interface. For perspective, a PC has a CPU, with the operating system running on the CPU. It has an Ethernet network interface card (NIC). The OS configuration includes an IP address associated with the NIC, either configured or learned dynamically with DHCP.

A switch uses the same ideas, except that the switch needs to use a virtual NIC inside the switch. Like a PC, a switch has a real CPU, running an OS (called IOS). The switch obviously has lots of Ethernet ports, but instead of assigning its management IP address to any of those ports, the switch then uses a NIC-like concept called a switched virtual interface (SVI), or more commonly, a VLAN interface, that acts like the switch's own NIC. Then the settings on the switch look something like a host, with the switch configuration assigning IP settings, like an IP address, to this VLAN interface, as shown in Figure 8-6.

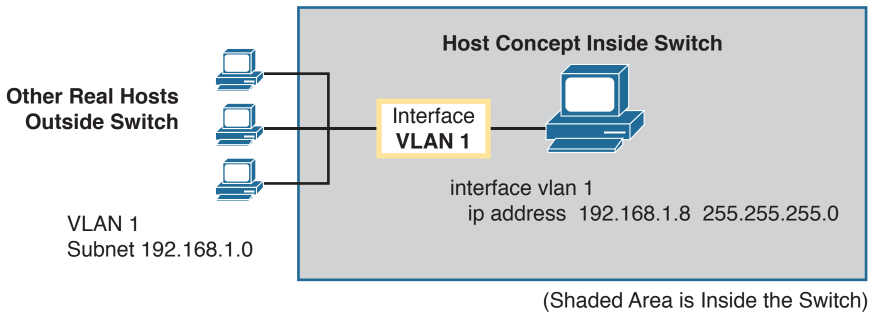

Host Concept Inside Switch
Other Real Hosts Outside Switch
Interface VLAN 1
interface vlan 1
ip address 192.168.1.8 255.255.255.0
VLAN 1
Subnet 192.168.1.0
(Shaded Area is Inside the Switch)

Figure 8-6 *Switch Virtual Interface (SVI) Concept Inside a Switch*

By using interface VLAN 1 for the IP configuration, the switch can then send and receive frames on any of the ports in VLAN 1. In a Cisco switch, by default, all ports are assigned to VLAN 1.

In most networks, switches configure many VLANs, so the network engineer has a choice of where to configure the IP address. That is, the management IP address does not have to be configured on the VLAN 1 interface (as configured with the **interface vlan 1** command seen in Figure 8-6).

A Layer 2 Cisco LAN switch often uses a single VLAN interface at a time, although multiple VLAN interfaces can be configured. The switch only needs one IP address for management purposes. But you can configure VLAN interfaces and assign them IP addresses for any working VLAN.

For example, Figure 8-7 shows a Layer 2 switch with some physical ports in two different VLANs (VLANs 1 and 2). The figure also shows the subnets used on those VLANs. The network engineer could choose to create a VLAN 1 interface, a VLAN 2 interface, or both. In most cases, the engineer plans which VLAN to use when managing a group of switches, and creates a VLAN interface for that VLAN only.

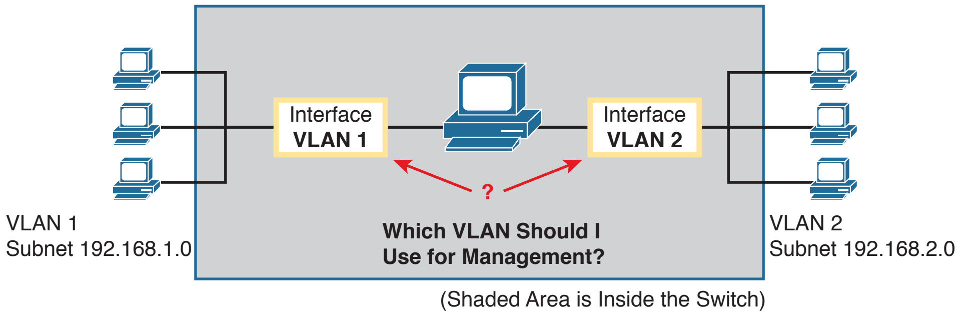


Figure 8-7 *Choosing One VLAN on Which to Configure a Switch IP Address*

Note that you should not try to use a VLAN interface for which there are no physical ports assigned to the same VLAN. If you do, the VLAN interface will not reach an up/up state, and the switch will not have the physical ability to communicate outside the switch.

NOTE Some Cisco switches can be configured to act as either a Layer 2 switch or a Layer 3 switch. When acting as a Layer 2 switch, a switch forwards Ethernet frames as discussed in depth in Chapter 7, "Analyzing Ethernet LAN Switching." Alternatively, a switch can also act as a *multilayer switch* or *Layer 3 switch*, which means the switch can do both Layer 2 switching and Layer 3 IP routing of IP packets, using the Layer 3 logic normally used by routers. This chapter assumes all switches are Layer 2 switches. Chapter 11, "Implementing Ethernet Virtual LANs," further defines the differences between these types of operation for LAN switches.

Configuring the IP address (and mask) on one VLAN interface allows the switch to send and receive IP packets with other hosts in a subnet that exists on that VLAN; however, the switch cannot communicate outside the local subnet without another configuration setting called the default gateway. The reason a switch needs a default gateway setting is the same reason that hosts need the same setting—because of how hosts think when sending IP packets. Specifically:

- To send IP packets to hosts in the same subnet, send them directly
- To send IP packets to hosts in a different subnet, send them to the local router; that is, the default gateway

Figure 8-8 shows the ideas. In this case, the switch (on the right) will use IP address 192.168.1.200 as configured on interface VLAN 1. However, to communicate with host A, on the far left of the figure, the switch must use Router R1 (the default gateway) to forward IP packets to host A. To make that work, the switch needs to configure a default gateway setting, pointing to Router R1's IP address (192.168.1.1 in this case). Note that the switch and router both use the same mask, 255.255.255.0, which puts the addresses in the same subnet.

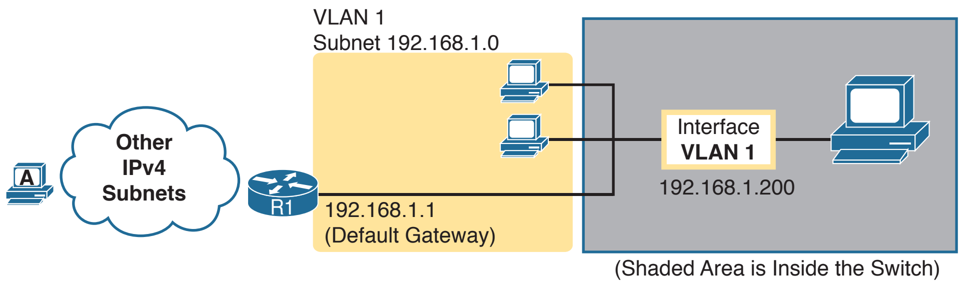


Figure 8-8 *The Need for a Default Gateway*

Configuring IPv4 on a Switch

A switch configures its IPv4 address and mask on this special NIC-like *VLAN interface*. The following steps list the commands used to configure IPv4 on a switch, assuming that the IP address is configured to be in VLAN 1, with Example 8-7 that follows showing an example configuration.

Config Checklist

Step 1. Use the **interface vlan 1** command in global configuration mode to enter interface VLAN 1 configuration mode.

Step 2. Use the **ip address** *ip-address mask* command in interface configuration mode to assign an IP address and mask.

Step 3. Use the **no shutdown** command in interface configuration mode to enable the VLAN 1 interface if it is not already enabled.

Step 4. Add the **ip default-gateway** *ip-address* command in global configuration mode to configure the default gateway.

Step 5. (Optional) Add the **ip name-server** *ip-address1 ip-address2* … command in global configuration mode to configure the switch to use Domain Name System (DNS) to resolve names into their matching IP address.

Example 8-7 *Switch Static IP Address Configuration*

```
Emma# configure terminal
Emma(config)# interface vlan 1
Emma(config-if)# ip address 192.168.1.200 255.255.255.0
Emma(config-if)# no shutdown
00:25:07: %LINK-3-UPDOWN: Interface Vlan1, changed state to up
00:25:08: %LINEPROTO-5-UPDOWN: Line protocol on Interface Vlan1, changed
  state to up
Emma(config-if)# exit
Emma(config)# ip default-gateway 192.168.1.1
```

On a side note, this example shows a particularly important and common command: the [**no**] **shutdown** command. To administratively enable an interface on a switch, use the **no shutdown** interface subcommand; to disable an interface, use the **shutdown** interface subcommand. This command can be used on the physical Ethernet interfaces that the switch uses to switch Ethernet messages in addition to the VLAN interface shown here in this example.

Also, pause long enough to look at the messages that appear just below the **no shutdown** command in Example 8-7. Those messages are syslog messages generated by the switch stating that the switch did indeed enable the interface. Switches (and routers) generate syslog messages in response to a variety of events, and by default, those messages appear at the console. Chapter 33, "Device Management Protocols," discusses syslog messages in more detail.

Configuring a Switch to Learn Its IP Address with DHCP

The switch can also use Dynamic Host Configuration Protocol (DHCP) to dynamically learn its IPv4 settings. Basically, all you have to do is tell the switch to use DHCP on the interface, and enable the interface. Assuming that DHCP works in this network, the switch will learn all its settings. The following list details the steps, again assuming the use of interface VLAN 1, with Example 8-8 that follows showing an example:

Step 1. Enter VLAN 1 configuration mode using the **interface vlan 1** global configuration command, and enable the interface using the **no shutdown** command as necessary.

Step 2. Assign an IP address and mask using the **ip address dhcp** interface subcommand.

Example 8-8 *Switch Dynamic IP Address Configuration with DHCP*

```
Emma# configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
Emma(config)# interface vlan 1
Emma(config-if)# ip address dhcp
Emma(config-if)# no shutdown
Emma(config-if)# ^Z
Emma#
00:38:20: %LINK-3-UPDOWN: Interface Vlan1, changed state to up
00:38:21: %LINEPROTO-5-UPDOWN: Line protocol on Interface Vlan1, changed state to up
```

Verifying IPv4 on a Switch

The switch IPv4 configuration can be checked in several places. First, you can always look at the current configuration using the **show running-config** command. Second, you can look at the IP address and mask information using the **show interfaces vlan** *x* command, which shows detailed status information about the VLAN interface in VLAN *x*. Finally, if using DHCP, use the **show dhcp lease** command to see the (temporarily) leased IP address and other parameters. (Note that the switch does not store the DHCP-learned IP configuration in the running-config file.) Example 8-9 shows sample output from these commands to match the configuration in Example 8-8.

Example 8-9 *Verifying DHCP-Learned Information on a Switch*

```
Emma# show dhcp lease
Temp IP addr: 192.168.1.101  for peer on Interface: Vlan1
Temp  sub net mask: 255.255.255.0
   DHCP Lease server: 192.168.1.1, state: 3 Bound
   DHCP transaction id: 1966
   Lease: 86400 secs,  Renewal: 43200 secs,  Rebind: 75600 secs
Temp default-gateway addr: 192.168.1.1
   Next timer fires after: 11:59:45
   Retry count: 0   Client-ID: cisco-0019.e86a.6fc0-Vl1
   Hostname: Emma
Emma# show interfaces vlan 1
Vlan1 is up, line protocol is up
  Hardware is EtherSVI, address is 0019.e86a.6fc0 (bia 0019.e86a.6fc0)
  Internet address is 192.168.1.101/24
  MTU 1500 bytes, BW 1000000 Kbit, DLY 10 usec,
     reliability 255/255, txload 1/255, rxload 1/255
! lines omitted for brevity
Emma# show ip default-gateway
192.168.1.1
```

The output of the **show interfaces vlan 1** command lists two very important details related to switch IP addressing. First, this **show** command lists the interface status of the VLAN 1 interface—in this case, “up and up.” If the VLAN 1 interface is not up, the switch cannot use its IP address to send and receive management traffic. Notably, if you forget to issue the **no shutdown** command, the VLAN 1 interface remains in its default shutdown state and is listed as "administratively down" in the **show** command output.

Second, note that the output lists the interface's IP address on the third line. If you statically configure the IP address, as in Example 8-7, the IP address will always be listed; however, if you use DHCP and DHCP fails, the **show interfaces vlan** *x* command will not list an IP address here. When DHCP works, you can see the IP address with the **show interfaces vlan 1** command, but that output does not remind you whether the address is either statically configured or DHCP leased. So it does take a little extra effort to make sure you know whether the address is statically configured or DHCP-learned on the VLAN interface.

Miscellaneous Settings Useful in Lab

This last short section of the chapter touches on a couple of commands that can help you be a little more productive when practicing in a lab.

History Buffer Commands

When you enter commands from the CLI, the switch saves the last several commands in the history buffer. Then, as mentioned in Chapter 6, "Using the Command-Line Interface," you can use the up-arrow key or press Ctrl+P to move back in the history buffer to retrieve a command you entered a few commands ago. This feature makes it very easy and fast to use a set of commands repeatedly. Table 8-2 lists some of the key commands related to the history buffer.

Table 8-2 Commands Related to the History Buffer

Command	Description
show history	An EXEC command that lists the commands currently held in the history buffer.
terminal history size *x*	From EXEC mode, this command allows a single user to set, just for this one login session, the size of his or her history buffer.
history size *x*	A configuration command that from console or vty line configuration mode, sets the default number of commands saved in the history buffer for the users of the console or vty lines, respectively.

The logging synchronous, exec-timeout, and no ip domain-lookup Commands

These next three configuration commands have little in common, other than the fact that they can be useful settings to reduce your frustration when using the console of a switch or router.

The console automatically receives copies of all unsolicited syslog messages on a switch. The idea is that if the switch needs to tell the network administrator some important and possibly urgent information, the administrator might be at the console and might notice the message.

Unfortunately, IOS (by default) displays these syslog messages on the console's screen at any time—including right in the middle of a command you are entering, or in the middle of the output of a **show** command. Having a bunch of text show up unexpectedly can be a bit annoying.

You could simply disable the feature that sends these messages to the console, and then re-enable the feature later, using the **no logging console** and **logging console** global commands. For example, when working from the console, if you want to temporarily not be bothered by log messages, you can disable the display of these messages with the **no logging console** global configuration command, and then when finished, enable them again.

However, IOS supplies a reasonable compromise, telling the switch to display syslog messages only at more convenient times, such as at the end of output from a **show** command. To do so, just configure the **logging synchronous** console line subcommand, which basically tells IOS to synchronize the syslog message display with the messages requested using **show** commands.

Another way to improve the user experience at the console is to control timeouts of the login session from the console or when using Telnet or SSH. By default, the switch automatically disconnects console and vty (Telnet and SSH) users after 5 minutes of inactivity. The **exec-timeout** *minutes seconds* line subcommand enables you to set the length of that inactivity timer. In lab (but not in production), you might want to use the special value of 0 minutes and 0 seconds meaning "never time out."

Finally, IOS has an interesting combination of features that can make you wait for a minute or so when you mistype a command. First, IOS tries to use DNS name resolution on IP hostnames—a generally useful feature. If you mistype a command, however, IOS thinks you want to Telnet to a host by that name. With all default settings in the switch, the switch tries to resolve the hostname, cannot find a DNS server, and takes about a minute to timeout and give you control of the CLI again.

To avoid this problem, configure the **no ip domain-lookup** global configuration command, which disables IOS's attempt to resolve the hostname into an IP address.

Example 8-10 collects all these commands into a single example, as a template for some good settings to add in a lab switch to make you more productive.

Example 8-10 *Commands Often Used in Lab to Increase Productivity*

```
no ip domain-lookup
!
line console 0
 exec-timeout 0 0
 logging synchronous
 history size 20
!
line vty 0 15
 exec-timeout 0 0
 logging synchronous
 history size 20
```

5. A switch's port Gi0/1 has been correctly enabled with port security. The configuration sets the violation mode to restrict. A frame that violates the port security policy enters the interface, followed by a frame that does not. Which of the following answers correctly describe what happens in this scenario? (Choose two answers.)

- **a.** The switch puts the interface into an err-disabled state when the first frame arrives.
- **b.** The switch generates syslog messages about the violating traffic for the first frame.
- **c.** The switch increments the violation counter for Gi0/1 by 1.
- **d.** The switch discards both the first and second frame.

6. A Cisco Catalyst switch connects to what should be individual user PCs. Each port has the same port security configuration, configured as follows:

```
interface range gigabitethernet 0/1 - 24
 switchport mode access
 switchport port-security
 switchport port-security mac-address sticky
```

Which of the following answers describe the result of the port security configuration created with these commands? (Choose two answers.)

- **a.** Prevents unknown devices with unknown MAC addresses from sending data through the switch ports.
- **b.** If a user connects a switch to the cable, prevents multiple devices from sending data through the port.
- **c.** Will allow any one device to connect to each port, and will save that device's MAC address into the startup-config.
- **d.** Will allow any one device to connect to each port, but *will not* save that device's MAC address into the startup-config.

Foundation Topics

Configuring Switch Interfaces

IOS uses the term *interface* to refer to physical ports used to forward data to and from other devices. Each interface can be configured with several settings, each of which might differ from interface to interface. IOS uses interface subcommands to configure these settings. Each of these settings may be different from one interface to the next, so you would first identify the specific interface, and then configure the specific setting.

This section begins with a discussion of three relatively basic per-interface settings: the port speed, duplex, and a text description. Following that, the text takes a short look at a pair of the most common interface subcommands: the **shutdown** and **no shutdown** commands, which administratively disable and enable the interface, respectively. This section ends with a discussion about autonegotiation concepts, which in turn dictates what settings a switch chooses to use when using autonegotiation.

NOTE If you would like more detail about Cisco recommendations about what to put in what VLAN, which impacts the size of VLANs, read the most recent Cisco document, "Campus LAN validated design" by searching on that phrase at Cisco.com.

Summarizing the main points about broadcast domains:

Key Topic

- Broadcasts exists, so be ready to analyze a design to define each broadcast domain, that is, each set of devices whose broadcasts reach the other devices in that domain.
- VLANs by definition are broadcast domains created though configuration.
- Routers, because they do not forward LAN broadcasts, create separate broadcast domains off their separate Ethernet interfaces.

Analyzing Campus LAN Topologies

The term *campus LAN* refers to the LAN created to support the devices in a building or in multiple buildings in somewhat close proximity to one another. For example, a company might lease office space in several buildings in the same office park. The network engineers can then build a campus LAN that includes switches in each building, plus Ethernet links between the switches in the buildings, to create a larger campus LAN.

When planning and designing a campus LAN, the engineers must consider the types of Ethernet available and the cabling lengths supported by each type. The engineers also need to choose the speeds required for each Ethernet segment. In addition, some thought needs to be given to the idea that some switches should be used to connect directly to end-user devices, whereas other switches might need to simply connect to a large number of these end-user switches. Finally, most projects require that the engineer consider the type of equipment that is already installed and whether an increase in speed on some segments is worth the cost of buying new equipment.

This second of three major sections of the chapter discusses the topology of a campus LAN design. Network designers do not just plug in devices to any port and connect switches to each other in an arbitrary way, like you might do with a few devices on the same table in a lab. Instead, there are known better ways to design the topology of a campus LAN, and this section introduces some of the key points and terms. The last major section of the chapter then looks at how to choose which Ethernet standard to use for each link in that campus LAN design, and why you might choose one versus another.

Two-Tier Campus Design (Collapsed Core)

To sift through all the requirements for a campus LAN, and then have a reasonable conversation about it with peers, most Cisco-oriented LAN designs use some common terminology to refer to the design. For this book's purposes, you should be aware of some of the key campus LAN design terminology.

The Two-Tier Campus Design

Figure 10-10 shows a typical design of a large campus LAN, with the terminology included in the figure. This LAN has around 1000 PCs connected to switches that support around 25 ports each. Explanations of the terminology follow the figure.

Key Topic

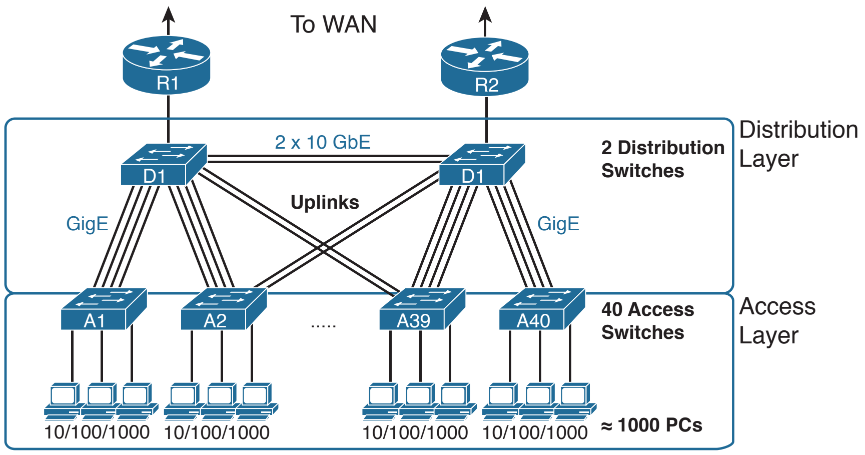


Figure 10-10 *Campus LAN with Design Terminology Listed*

Cisco uses three terms to describe the role of each switch in a campus design: *access*, *distribution*, and *core*. The roles differ based on whether the switch forwards traffic from user devices and the rest of the LAN (access), or whether the switch forwards traffic between other LAN switches (distribution and core).

Access switches connect directly to end users, providing user device access to the LAN. Access switches normally send traffic to and from the end-user devices to which they are connected and sit at the edge of the LAN.

Distribution switches provide a path through which the access switches can forward traffic to each other. By design, each of the access switches connects to at least one distribution switch, typically to two distribution switches for redundancy. The distribution switches provide the service of forwarding traffic to other parts of the LAN. Note that most designs use at least two uplinks to two different distribution switches (as shown in Figure 10-10) for redundancy.

The figure shows a two-tier design, with the tiers being the access tier (or layer) and the distribution tier (or layer). A two-tier design solves two major design needs:

- Provides a place to connect end-user devices (the access layer, with access switches)
- Connects the switches with a reasonable number of cables and switch ports by connecting all 40 access switches to two distribution switches

Topology Terminology Seen Within a Two-Tier Design

The exam topics happen to list a couple of terms about LAN and WAN topology and design, so this is a good place to pause to discuss those terms for a moment.

First, consider these more formal definitions of four topology terms:

Key Topic

Star: A design in which one central device connects to several others, so that if you drew the links out in all directions, the design would look like a star with light shining in all directions.

Full mesh: For any set of network nodes, a design that connects a link between each pair of nodes.

Partial mesh: For any set of network nodes, a design that connects a link between some pairs of nodes, but not all. In other words, a mesh that is not a full mesh.

Hybrid: A design that combines topology design concepts into a larger (typically more complex) design.

Armed with those formal definitions, note that the two-tier design is indeed a hybrid design that uses both a star topology at the access layer and a partial mesh at the distribution layer. To see why, consider Figure 10-11. It redraws a typical access layer switch, but instead of putting the PCs all below the switch, it spreads them around the switch. Then on the right, a similar version of the same drawing shows why the term star might be used—the topology looks a little like a child's drawing of a star.

Key Topic

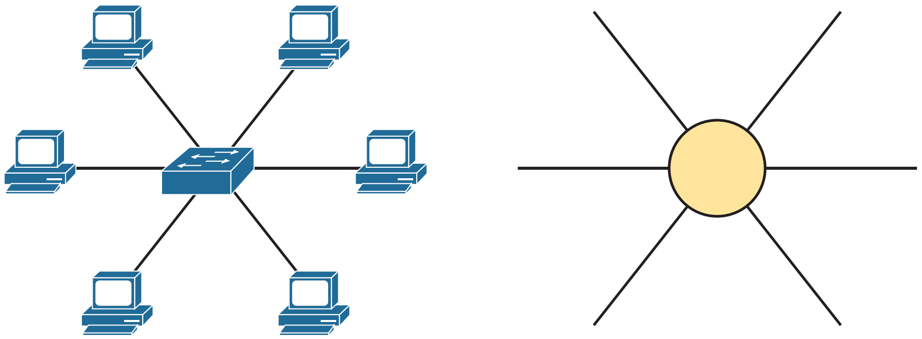


Figure 10-11 *The Star Topology Design Concept in Networking*

The distribution layer creates a partial mesh. If you view the access and distribution switches as nodes in a design, some nodes have a link between them, and some do not. Just refer to Figure 10-10 and note that, by design, none of the access layer switches connect to each other.

Finally, a design could use a full mesh. However, for a variety of reasons beyond the scope of the design discussion here, a campus design typically does not need to use the number of links and ports required by a full mesh design. However, just to make the point, first consider how many links and switch ports would be required for a single link between nodes in a full mesh, with six nodes, as shown in Figure 10-12.

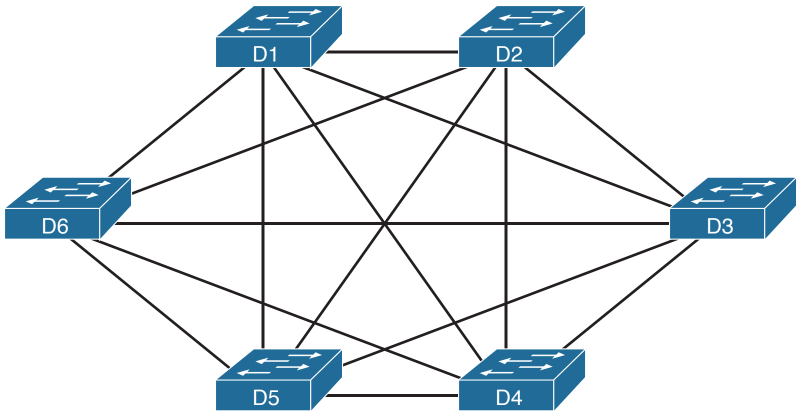


Figure 10-12 *Using a Full Mesh at the Distribution Layer, 6 Switches, 15 Links*

Even with only six switches, a full mesh would consume 15 links (and 30 switch ports—two per link).

Now think about a full mesh at the distribution layer for a design like Figure 10-10, with 40 access switches and two distribution switches. Rather than drawing it and counting it, the number of links is calculated with this old math formula from high school: $N(N - 1) / 2$, or in this case, 42 * 41 / 2 = 861 links, and 1722 switch ports consumed among all switches.

For comparison's sake, the partial mesh design of Figure 10-10, with a pair of links from each access switch to each distribution switch, requires only 160 links and a total of 320 ports among all switches.

Three-Tier Campus Design (Core)

The two-tier design of Figure 10-10, with a partial mesh of links at the distribution layer, happens to be the most common campus LAN design. It also goes by two common names: a two-tier design (for obvious reasons), and a collapsed core (for less obvious reasons). The term *collapsed core* refers to the fact that the two-tier design does not have a third tier, the core tier. This next topic examines a three-tier design that does have a core, for perspective.

Imagine your campus has just two or three buildings. Each building has a two-tier design inside the building, with a pair of distribution switches in each building and access switches spread around the building as needed. How would you connect the LANs in each building? Well, with just a few buildings, it makes sense to simply cable the distribution switches together, as shown in Figure 10-13.

Key Topic

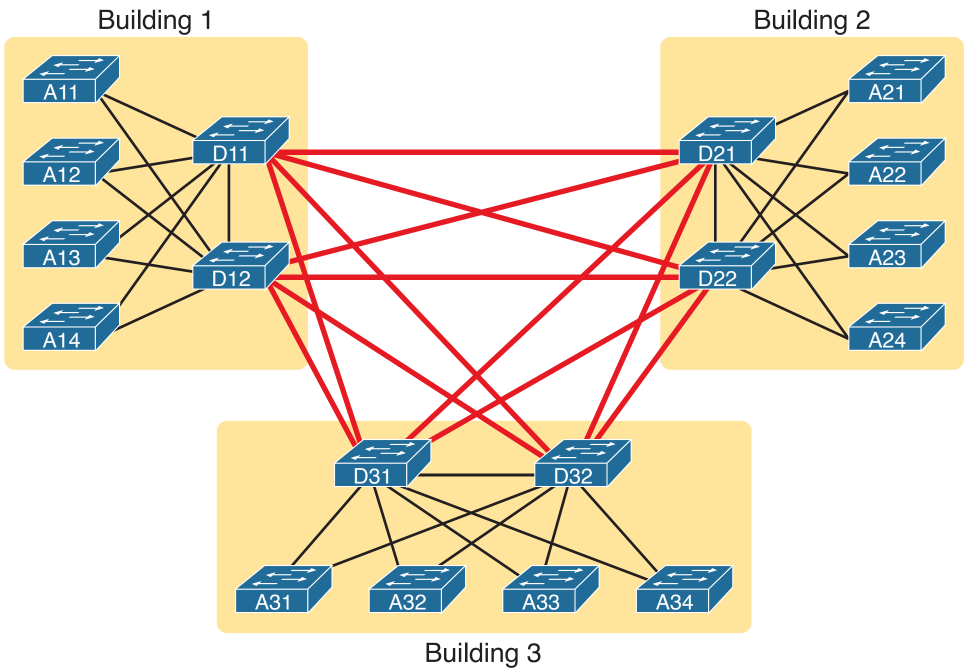


Figure 10-13 *Two-Tier Building Design, No Core, Three Buildings*

The design in Figure 10-13 works well, and many companies use this design. Sometimes the center of the network uses a full mesh, sometimes a partial mesh, depending on the availability of cables between the buildings.

However, a design with a third tier (a core tier) saves on switch ports and on cables in larger designs. And note that with the links between buildings, the cables run outside, are often more expensive to install, are almost always fiber cabling with more expensive switch ports, so conserving the number of cables used between buildings can help reduce costs.

A three-tier core design, unsurprisingly at this point, adds a few more switches (core switches), which provide one function: to connect the distribution switches. Figure 10-14 shows the migration of the Figure 10-13 collapsed core (that is, a design without a core) to a three-tier core design.

Key Topic

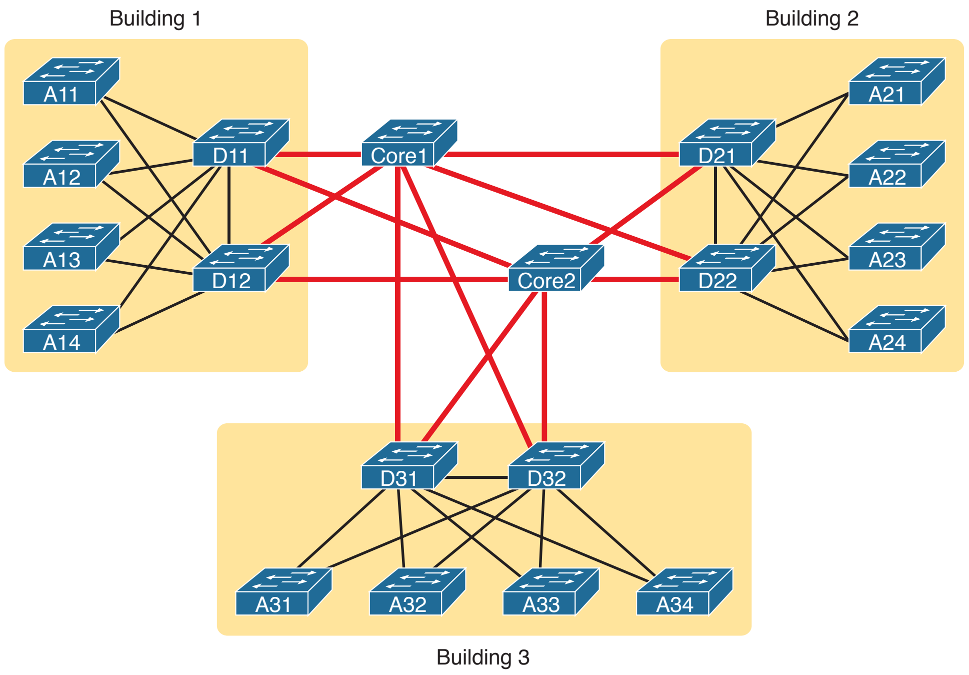


Figure 10-14 *Three-Tier Building Design (Core Design), Three Buildings*

NOTE The core switches sit in the middle of the figure. In the physical world, they often sit in the same room as one of the distribution switches, rather than in some purpose-built room in the middle of the office park. The figure focuses more on the topology rather than the physical location.

By using a core design, with a partial mesh of links in the core, you still provide connectivity to all parts of the LAN, and to the routers that send packets over the WAN, just with fewer links between buildings.

The following list summarizes the terms that describe the roles of campus switches:

- **Access:** Provides a connection point (access) for end-user devices. Does not forward frames between two other access switches under normal circumstances.
- **Distribution:** Provides an aggregation point for access switches, providing connectivity to the rest of the devices in the LAN, forwarding frames between switches, but not connecting directly to end-user devices.
- **Core:** Aggregates distribution switches in very large campus LANs, providing very high forwarding rates for the larger volume of traffic due to the size of the network.

Topology Design Terminology

The ICND1 and CCNA exam topics specifically mention several network design terms related to topology. This next topic summarizes those key terms to connect the terms to the matching ideas.

First, consider Figure 10-15, which shows a few of the terms. First, on the left, drawings often show access switches with a series of cables, parallel to each other. However, an access switch and its access links is often called a *star topology*. Why? Look at the redrawn access switch in the center of the figure, with the cables radiating out from the center. It does not look like a real star, but it looks a little like a child's drawing of a star, hence the term star topology.

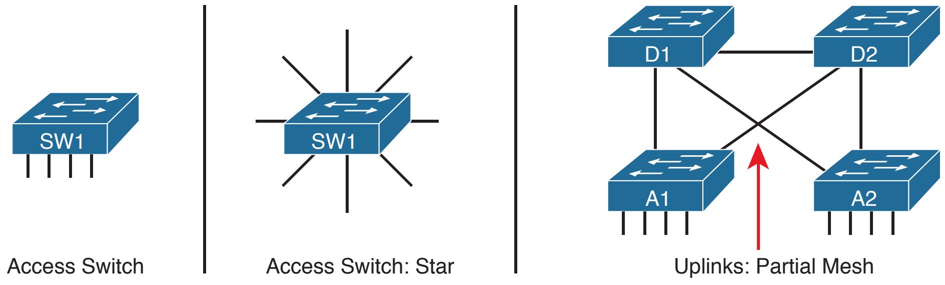


Figure 10-15 *LAN Design Terminology*

The right side of the figure repeats a typical two-tier design, focusing on the mesh of links between the access and distribution switches. Any group of nodes that connect with more links than a star topology is typically called a *mesh*. In this case, the mesh is a *partial mesh*, because not all nodes have a direct link between each other. A design that connects all nodes with a link would be a *full mesh*.

Real networks make use of these topology ideas, but often a network combines the ideas together. For instance, the right side of Figure 10-14 combines the star topology of the access layer with the partial mesh of the distribution layer. So you might hear these designs that combine concepts called a *hybrid design*.

Analyzing LAN Physical Standard Choices

When you look at the design of a network designed by someone else, you can look at all the different types of cabling used, the different types of switch ports, and the Ethernet standards used in each case. Then ask yourself: Why did they choose a particular type of Ethernet link for each link in the network? Asking that question, and investigating the answer, starts to reveal much about building the physical campus LAN.

10

The IEEE has done an amazing job developing Ethernet standards that give network designers many options. Two themes in particular have helped Ethernet grow over the long term:

Key Topic

- The IEEE has developed many additional 802.3 standards for different types of cabling, different cable lengths, and for faster speeds.
- All the physical standards rely on the same consistent data-link details, with the same standard frame formats. That means that one Ethernet LAN can use many types of physical links to meet distance, budget, and cabling needs.

For example, think about the access layer of the generic design drawings, but now think about cabling and Ethernet standards. In practice, access layer switches sit in a locked wiring closet somewhere on the same floor as the end user devices. Electricians have installed unshielded twisted-pair (UTP) cabling used at the access layer, running from that wiring closet to each wall plate at each office, cubicle, or any place where an Ethernet device might need to connect to the LAN. The type and quality of the cabling installed

5. A switch has just arrived from Cisco. The switch has never been configured with any VLANs, but VTP has been disabled. An engineer gets into configuration mode and issues the **vlan 22** command, followed by the **name Hannahs-VLAN** command. Which of the following are true? (Choose two answers.)

- **a.** VLAN 22 is listed in the output of the **show vlan brief** command.
- **b.** VLAN 22 is listed in the output of the **show running-config** command.
- **c.** VLAN 22 is not created by this process.
- **d.** VLAN 22 does not exist in that switch until at least one interface is assigned to that VLAN.

6. Which of the following commands identify switch interfaces as being trunking interfaces: interfaces that currently operate as VLAN trunks? (Choose two answers.)

- **a.** **show interfaces**
- **b.** **show interfaces switchport**
- **c.** **show interfaces trunk**
- **d.** **show trunks**

Foundation Topics

Virtual LAN Concepts

Before understanding VLANs, you must first have a specific understanding of the definition of a LAN. For example, from one perspective, a LAN includes all the user devices, servers, switches, routers, cables, and wireless access points in one location. However, an alternative narrower definition of a LAN can help in understanding the concept of a virtual LAN:

A LAN includes all devices in the same broadcast domain.

A broadcast domain includes the set of all LAN-connected devices, so that when any of the devices sends a broadcast frame, all the other devices get a copy of the frame. So, from one perspective, you can think of a LAN and a broadcast domain as being basically the same thing.

Without VLANs, a switch considers all its interfaces to be in the same broadcast domain. That is, for one switch, when a broadcast frame entered one switch port, the switch forwarded that broadcast frame out all other ports. With that logic, to create two different LAN broadcast domains, you had to buy two different Ethernet LAN switches, as shown in Figure 11-1.

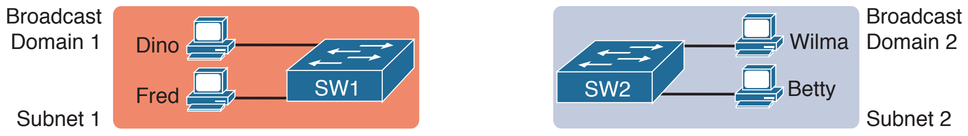


Figure 11-1 *Creating Two Broadcast Domains with Two Physical Switches and No VLANs*

With support for VLANs, a single switch can accomplish the same goals of the design in Figure 11-1—to create two broadcast domains—with a single switch. With VLANs, a switch can configure some interfaces into one broadcast domain and some into another, creating multiple broadcast domains. These individual broadcast domains created by the switch are called virtual LANs (VLAN).

For example, in Figure 11-2, the single switch creates two VLANs, treating the ports in each VLAN as being completely separate. The switch would never forward a frame sent by Dino (in VLAN 1) over to either Wilma or Betty (in VLAN 2).

Key Topic

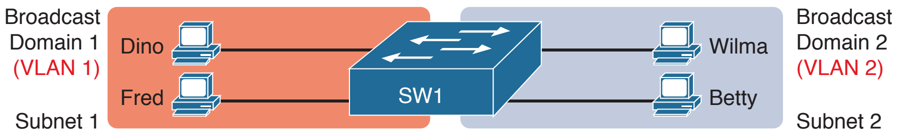


Figure 11-2 *Creating Two Broadcast Domains Using One Switch and VLANs*

Designing campus LANs to use more VLANs, each with a smaller number of devices, often helps improve the LAN in many ways. For example, a broadcast sent by one host in a VLAN will be received and processed by all the other hosts in the VLAN—but not by hosts in a different VLAN. Limiting the number of hosts that receive a single broadcast frame reduces the number of hosts that waste effort processing unneeded broadcasts. It also reduces security risks, because fewer hosts see frames sent by any one host. These are just a few reasons for separating hosts into different VLANs. The following list summarizes the most common reasons for choosing to create smaller broadcast domains (VLANs):

Key Topic

- To reduce CPU overhead on each device by reducing the number of devices that receive each broadcast frame
- To reduce security risks by reducing the number of hosts that receive copies of frames that the switches flood (broadcasts, multicasts, and unknown unicasts)
- To improve security for hosts that send sensitive data by keeping those hosts on a separate VLAN
- To create more flexible designs that group users by department, or by groups that work together, instead of by physical location
- To solve problems more quickly, because the failure domain for many problems is the same set of devices as those in the same broadcast domain
- To reduce the workload for the Spanning Tree Protocol (STP) by limiting a VLAN to a single access switch

This chapter does not examine all the reasons for VLANs in more depth. However, know that most enterprise networks use VLANs quite a bit. The rest of this chapter looks closely

Answers to the "Do I Know This Already?" quiz:

1 B **2** D **3** B **4** A, C **5** A, B **6** B, C at the mechanics of how VLANs work across multiple Cisco switches, including the required configuration. To that end, the next section examines VLAN trunking, a feature required when installing a VLAN that exists on more than one LAN switch.

Creating Multiswitch VLANs Using Trunking

Configuring VLANs on a single switch requires only a little effort: You simply configure each port to tell it the VLAN number to which the port belongs. With multiple switches, you have to consider additional concepts about how to forward traffic between the switches.

When using VLANs in networks that have multiple interconnected switches, the switches need to use *VLAN trunking* on the links between the switches. VLAN trunking causes the switches to use a process called *VLAN tagging*, by which the sending switch adds another header to the frame before sending it over the trunk. This extra trunking header includes a *VLAN identifier* (VLAN ID) field so that the sending switch can associate the frame with a particular VLAN ID, and the receiving switch can then know in what VLAN each frame belongs.

Figure 11-3 shows an example that demonstrates VLANs that exist on multiple switches, but it does not use trunking. First, the design uses two VLANs: VLAN 10 and VLAN 20. Each switch has two ports assigned to each VLAN, so each VLAN exists in both switches. To forward traffic in VLAN 10 between the two switches, the design includes a link between switches, with that link fully inside VLAN 10. Likewise, to support VLAN 20 traffic between switches, the design uses a second link between switches, with that link inside VLAN 20.

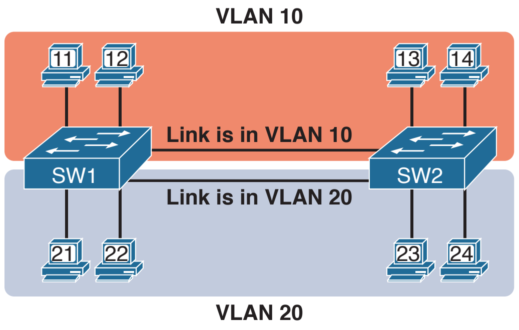


Figure 11-3 *Multiswitch VLAN Without VLAN Trunking*

The design in Figure 11-3 functions perfectly. For example, PC11 (in VLAN 10) can send a frame to PC14. The frame flows into SW1, over the top link (the one that is in VLAN 10) and over to SW2.

The design shown in Figure 11-3 works, but it simply does not scale very well. It requires one physical link between switches to support every VLAN. If a design needed 10 or 20 VLANs, you would need 10 or 20 links between switches, and you would use 10 or 20 switch ports (on each switch) for those links.

VLAN Tagging Concepts

VLAN trunking creates one link between switches that supports as many VLANs as you need. As a VLAN trunk, the switches treat the link as if it were a part of all the VLANs. At the same time, the trunk keeps the VLAN traffic separate, so frames in VLAN 10 would not go to devices in VLAN 20, and vice versa, because each frame is identified by VLAN number as it crosses the trunk. Figure 11-4 shows the idea, with a single physical link between the two switches.

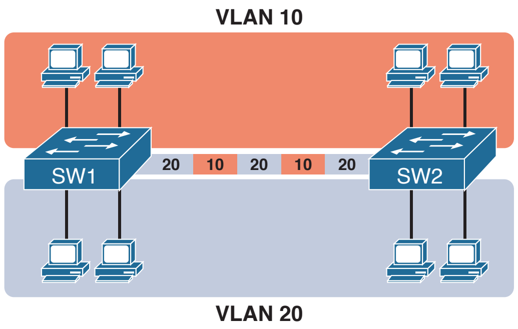


Figure 11-4 *Multiswitch VLAN with Trunking*

The use of trunking allows switches to pass frames from multiple VLANs over a single physical connection by adding a small header to the Ethernet frame. For example, Figure 11-5 shows PC11 sending a broadcast frame on interface Fa0/1 at Step 1. To flood the frame, switch SW1 needs to forward the broadcast frame to switch SW2. However, SW1 needs to let SW2 know that the frame is part of VLAN 10, so that after the frame is received, SW2 will flood the frame only into VLAN 10, and not into VLAN 20. So, as shown at Step 2, before sending the frame, SW1 adds a VLAN header to the original Ethernet frame, with the VLAN header listing a VLAN ID of 10 in this case.

Key Topic

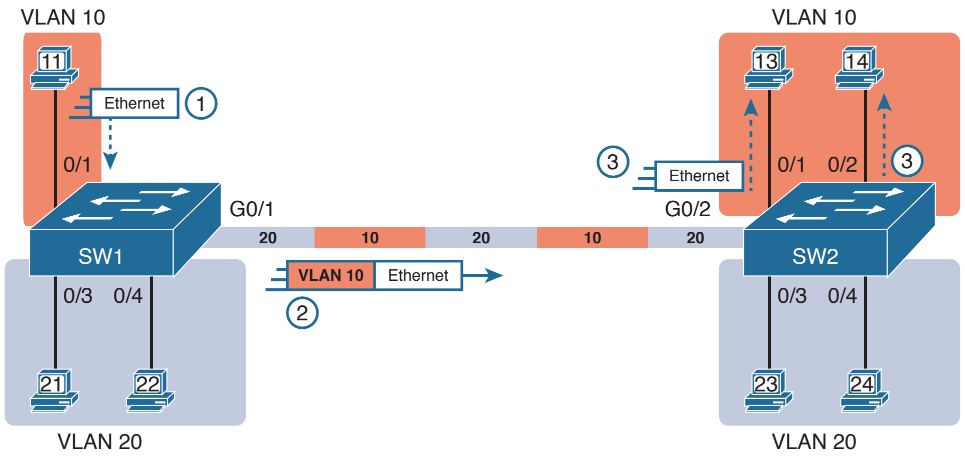


Figure 11-5 *VLAN Trunking Between Two Switches*

When SW2 receives the frame, it understands that the frame is in VLAN 10. SW2 then removes the VLAN header, forwarding the original frame out its interfaces in VLAN 10 (Step 3).

For another example, consider the case when PC21 (in VLAN 20) sends a broadcast. SW1 sends the broadcast out port Fa0/4 (because that port is in VLAN 20) and out Gi0/1 (because it is a trunk, meaning that it supports multiple different VLANs). SW1 adds a trunking header to the frame, listing a VLAN ID of 20. SW2 strips off the trunking header after determining that the frame is part of VLAN 20, so SW2 knows to forward the frame out only ports Fa0/3 and Fa0/4, because they are in VLAN 20, and not out ports Fa0/1 and Fa0/2, because they are in VLAN 10.

The 802.1Q and ISL VLAN Trunking Protocols

Cisco has supported two different trunking protocols over the years: Inter-Switch Link (ISL) and IEEE 802.1Q. Cisco created the ISL long before 802.1Q, in part because the IEEE had not yet defined a VLAN trunking standard. Years later, the IEEE completed work on the 802.1Q standard, which defines a different way to do trunking. Today, 802.1Q has become the more popular trunking protocol, with Cisco not even supporting ISL in some of its newer models of LAN switches, including the 2960 switches used in the examples in this book.

While both ISL and 802.1Q tag each frame with the VLAN ID, the details differ. 802.1Q inserts an extra 4-byte 802.1Q VLAN header into the original frame's Ethernet header, as shown at the top of Figure 11-6. As for the fields in the 802.1Q header, only the 12-bit VLAN ID field inside the 802.1Q header matters for topics discussed in this book. This 12-bit field supports a theoretical maximum of 2^{12} (4096) VLANs, but in practice it supports a maximum of 4094. (Both 802.1Q and ISL use 12 bits to tag the VLAN ID, with two reserved values [0 and 4095].)

Key Topic

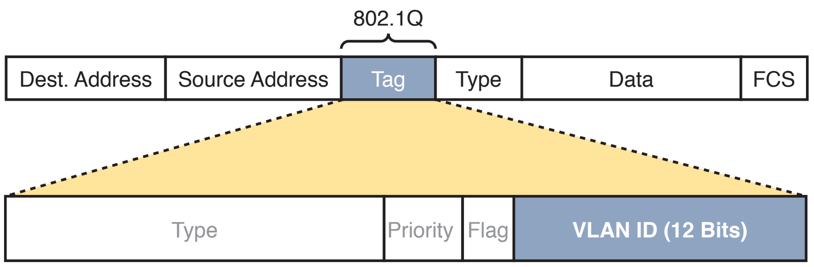


Figure 11-6 *802.1Q Trunking*

Cisco switches break the range of VLAN IDs (1–4094) into two ranges: the normal range and the extended range. All switches can use normal-range VLANs with values from 1 to 1005. Only some switches can use extended-range VLANs with VLAN IDs from 1006 to 4094. The rules for which switches can use extended-range VLANs depend on the configuration of the VLAN Trunking Protocol (VTP), which is discussed briefly in the section "VLAN Trunking Configuration," later in this chapter.

802.1Q also defines one special VLAN ID on each trunk as the *native VLAN* (defaulting to use VLAN 1). By definition, 802.1Q simply does not add an 802.1Q header to frames in the native VLAN. When the switch on the other side of the trunk receives a frame that does not have an 802.1Q header, the receiving switch knows that the frame is part of the native VLAN. Note that because of this behavior, both switches must agree on which VLAN is the native VLAN.

The 802.1Q native VLAN provides some interesting functions, mainly to support connections to devices that do not understand trunking. For example, a Cisco switch could be cabled to a switch that does not understand 802.1Q trunking. The Cisco switch could send frames in the native VLAN—meaning that the frame has no trunking header—so that the other switch would understand the frame. The native VLAN concept gives switches the capability of at least passing traffic in one VLAN (the native VLAN), which can allow some basic functions, like reachability to telnet into a switch.

Forwarding Data Between VLANs

If you create a campus LAN that contains many VLANs, you typically still need all devices to be able to send data to all other devices. This next topic discusses some concepts about how to route data between those VLANs.

First, it helps to know a few terms about some categories of LAN switches. All the Ethernet switch functions described in this book so far use the details and logic defined by OSI Layer 2 protocols. For example, Chapter 7, "Analyzing Ethernet LAN Switching," discussed how LAN switches receive Ethernet frames (a Layer 2 concept), look at the destination Ethernet MAC address (a Layer 2 address), and forward the Ethernet frame out some other interface. This chapter has already discussed the concept of VLANs as broadcast domains, which is yet another Layer 2 concept.

While some LAN switches work just as described so far in this book, some LAN switches have even more functions. LAN switches that forward data based on Layer 2 logic, as discussed so far in this book, often go by the name *Layer 2 switch*. However, some other switches can do some functions like a router, using additional logic defined by Layer 3 protocols. These switches go by the name *multilayer switch*, or *Layer 3 switch*. This section first discusses how to forward data between VLANs when using Layer 2 switches and ends with a brief discussion of how to use Layer 3 switches.

Routing Packets Between VLANs with a Router

When including VLANs in a campus LAN design, the devices in a VLAN need to be in the same subnet. Following the same design logic, devices in different VLANs need to be in different subnets. For example, in Figure 11-7, the two PCs on the left sit in VLAN 10, in subnet 10. The two PCs on the right sit in a different VLAN (20), with a different subnet (20).

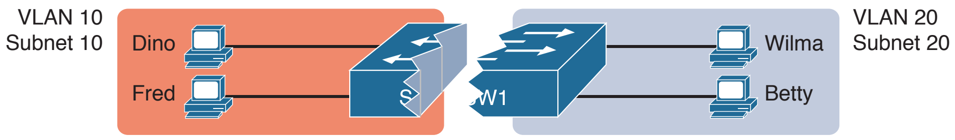


Figure 11-7 *Layer 2 Switch Does Not Route Between the VLANs*

NOTE The figure refers to subnets somewhat generally, like "subnet 10," just so the subnet numbers do not distract. Also, note that the subnet numbers do not have to be the same number as the VLAN numbers.

Figure 11-7 shows the switch as if it were two switches broken in two to emphasize the point that Layer 2 switches will not forward data between two VLANs. When configured with some ports in VLAN 10 and others in VLAN 20, the switch acts like two separate switches in which it will forward traffic. In fact, one goal of VLANs is to separate traffic in one VLAN from another, preventing frames in one VLAN from leaking over to other

VLANs. For example, when Dino (in VLAN 10) sends any Ethernet frame, if SW1 is a Layer 2 switch, that switch will not forward the frame to the PCs on the right in VLAN 20.

The network as a whole needs to support traffic flowing into and out of each VLAN, even though the Layer 2 switch does not forward frames outside a VLAN. The job of forwarding data into and out of a VLAN falls to routers. Instead of switching Layer 2 Ethernet frames between the two VLANs, the network must route Layer 3 packets between the two subnets.

That previous paragraph has some very specific wording related to Layers 2 and 3, so take a moment to reread and reconsider it for a moment. The Layer 2 logic does not let the Layer 2 switch forward the Layer 2 protocol data unit (L2PDU), the Ethernet frame, between VLANs. However, routers can route Layer 3 PDUs (L3PDU) (packets) between subnets as their normal job in life.

For example, Figure 11-8 shows a router that can route packets between subnets 10 and 20. The figure shows the same Layer 2 switch as shown in Figure 11-7, with the same perspective of the switch being split into parts with two different VLANs, and with the same PCs in the same VLANs and subnets. Now Router R1 has one LAN physical interface connected to the switch and assigned to VLAN 10, and a second physical interface connected to the switch and assigned to VLAN 20. With an interface connected to each subnet, the Layer 2 switch can keep doing its job—forwarding frames inside a VLAN, while the router can do its job—routing IP packets between the subnets.

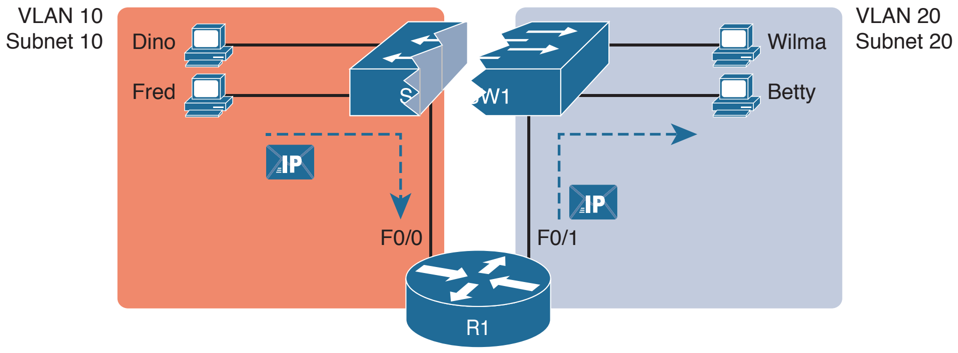


Figure 11-8 *Routing Between Two VLANs on Two Physical Interfaces*

The figure shows an IP packet being routed from Fred, which sits in one VLAN/subnet, to Betty, which sits in the other. The Layer 2 switch forwards two different Layer 2 Ethernet frames: one in VLAN 10, from Fred to R1's F0/0 interface, and the other in VLAN 20, from R1's F0/1 interface to Betty. From a Layer 3 perspective, Fred sends the IP packet to its default router (R1), and R1 routes the packet out another interface (F0/1) into another subnet where Betty resides.

While the design shown in Figure 11-8 works, it uses too many physical interfaces, one per VLAN. A much less expensive (and much preferred) option uses a VLAN trunk between the switch and router, requiring only one physical link between the router and switch, while supporting all VLANs. Trunking can work between any two devices that choose to support it: between two switches, between a router and a switch, or even between server hardware and a switch.

Figure 11-9 shows the same design idea as Figure 11-8, with the same packet being sent from Fred to Betty, except now R1 uses VLAN trunking instead of a separate link for each VLAN.

Key Topic

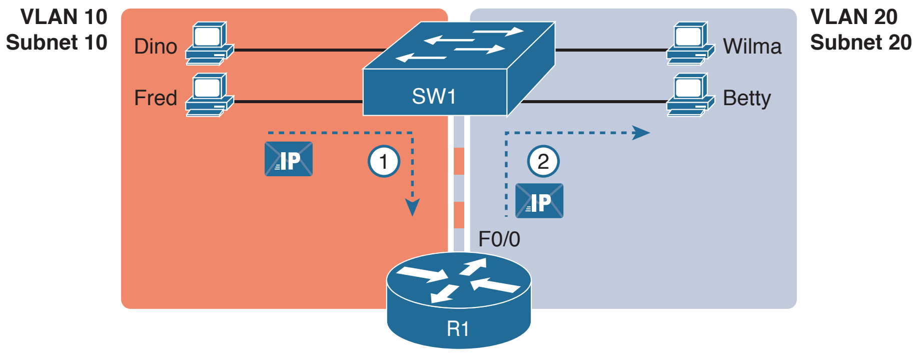


Figure 11-9 *Routing Between Two VLANs Using a Trunk on the Router*

NOTE Because the router has a single physical link connected to the LAN switch, this design is sometimes called a *router-on-a-stick*.

As a brief aside about terminology, many people describe the concept in Figures 11-8 and 11-9 as "routing packets between VLANs." You can use that phrase, and people know what you mean. However, note that this phrase is not literally true, because it refers to routing packets (a Layer 3 concept) and VLANs (a Layer 2 concept). It just takes fewer words to say something like "routing between VLANs" rather than the literally true but long "routing Layer 3 packets between Layer 3 subnets, with those subnets each mapping to a Layer 2 VLAN."

Routing Packets with a Layer 3 Switch

Routing packets using a physical router, even with the VLAN trunk in the router-on-a-stick model shown in Figure 11-9, still has one significant problem: performance. The physical link puts an upper limit on how many bits can be routed, and less expensive routers tend to be less powerful, and might not be able to route a large enough number of packets per second (pps) to keep up with the traffic volumes.

The ultimate solution moves the routing functions inside the LAN switch hardware. Vendors long ago started combining the hardware and software features of their Layer 2 LAN switches, plus their Layer 3 routers, creating products called *Layer 3 switches* (also known as *multilayer switches*). Layer 3 switches can be configured to act only as a Layer 2 switch, or they can be configured to do both Layer 2 switching as well as Layer 3 routing.

Today, many medium- to large-sized enterprise campus LANs use Layer 3 switches to route packets between subnets (VLANs) in a campus.

In concept, a Layer 3 switch works a lot like the original two devices on which the Layer 3 switch is based: a Layer 2 LAN switch and a Layer 3 router. In fact, if you take the concepts and packet flow shown in Figure 11-8, with a separate Layer 2 switch and Layer 3 router, and then imagine all those features happening inside one device, you have the general idea of what a Layer 3 switch does. Figure 11-10 shows that exact concept, repeating many details of Figure 11-8, but with an overlay that shows the one Layer 3 switch doing the Layer 2 switch functions and the separate Layer 3 routing function.

Key Topic

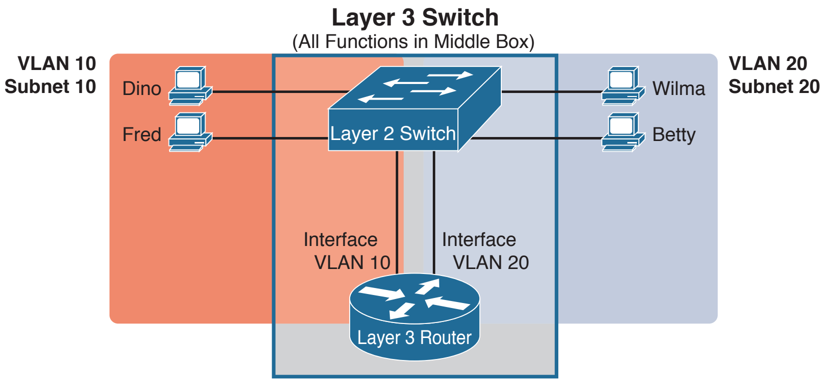


Figure 11-10 *Multilayer Switch: Layer 2 Switching with Layer 3 Routing in One Device*

This chapter introduces the core concepts of routing IP packets between VLANs (or more accurately, between the subnets on the VLANs). Chapter 18, "Configuring IPv4 Addresses and Static Routes," shows how to configure designs that use an external router with router-on-a-stick. This chapter now turns its attention to configuration and verification tasks for VLANs and VLAN trunks.

VLAN and VLAN Trunking Configuration and Verification

Cisco switches do not require any configuration to work. You can purchase Cisco switches, install devices with the correct cabling, turn on the switches, and they work. You would never need to configure the switch, and it would work fine, even if you interconnected switches, until you needed more than one VLAN. But if you want to use VLANs—and most enterprise networks do—you need to add some configuration.

This chapter separates the VLAN configuration details into two major sections. The first section looks at how to configure access interfaces, which are switch interfaces that do not use VLAN trunking. The second part shows how to configure interfaces that do use VLAN trunking.

Creating VLANs and Assigning Access VLANs to an Interface

This section shows how to create a VLAN, give the VLAN a name, and assign interfaces to a VLAN. To focus on these basic details, this section shows examples using a single switch, so VLAN trunking is not needed.

For a Cisco switch to forward frames in a particular VLAN, the switch must be configured to believe that the VLAN exists. In addition, the switch must have nontrunking interfaces (called *access interfaces*) assigned to the VLAN, and/or trunks that support the VLAN. The configuration steps for access interfaces are as follows, with the trunk configuration shown later in the section "VLAN Trunking Configuration":

Config Checklist

Step 1. To configure a new VLAN, follow these steps:

A. From configuration mode, use the **vlan** *vlan-id* command in global configuration mode to create the VLAN and to move the user into VLAN configuration mode.

B. (Optional) Use the **name** *name* command in VLAN configuration mode to list a name for the VLAN. If not configured, the VLAN name is VLAN*ZZZZ*, where *ZZZZ* is the four-digit decimal VLAN ID.

Step 2. For each access interface (each interface that does not trunk, but instead belongs to a single VLAN), follow these steps:

A. Use the **interface** *type number* command in global configuration mode to move into interface configuration mode for each desired interface.

B. Use the **switchport access vlan** *id-number* command in interface configuration mode to specify the VLAN number associated with that interface.

C. (Optional) Use the **switchport mode access** command in interface configuration mode to make this port always operate in access mode (that is, to not trunk).

While the list might look a little daunting, the process on a single switch is actually pretty simple. For example, if you want to put the switch's ports in three VLANs—11, 12, and 13—you just add three **vlan** commands: **vlan 11**, **vlan 12**, and **vlan 13**. Then, for each interface, add a **switchport access vlan 11** (or **12** or **13**) command to assign that interface to the proper VLAN.

NOTE The term *default VLAN* (as shown in the exam topics) refers to the default setting on the **switchport access vlan** *vlan-id* command, and that default is VLAN ID 1. In other words, by default, each port is assigned to access VLAN 1.

VLAN Configuration Example 1: Full VLAN Configuration

Example 11-1 shows the configuration process of adding a new VLAN and assigning access interfaces to that VLAN. Figure 11-11 shows the network used in the example, with one LAN switch (SW1) and two hosts in each of three VLANs (1, 2, and 3). The example shows the details of the two-step process for VLAN 2 and the interfaces in VLAN 2, with the configuration of VLAN 3 deferred until the next example.

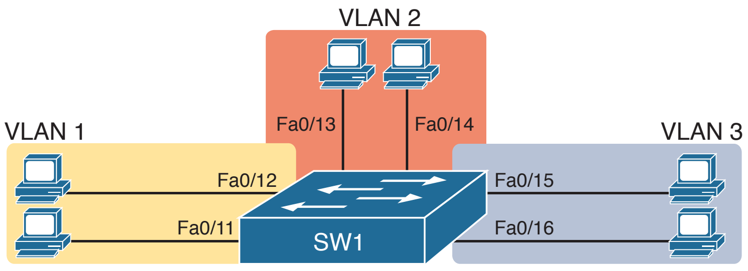


Figure 11-11 *Network with One Switch and Three VLANs*

Example 11-1 *Configuring VLANs and Assigning VLANs to Interfaces*

```
SW1# show vlan brief
VLAN Name                             Status    Ports
---- -------------------------------- --------- -------------------------------
1    default                          active    Fa0/1, Fa0/2, Fa0/3, Fa0/4
                                                Fa0/5, Fa0/6, Fa0/7, Fa0/8
                                                Fa0/9, Fa0/10, Fa0/11, Fa0/12
                                                Fa0/13, Fa0/14, Fa0/15, Fa0/16
                                                Fa0/17, Fa0/18, Fa0/19, Fa0/20
                                                Fa0/21, Fa0/22, Fa0/23, Fa0/24
                                                Gi0/1, Gi0/2
1002 fddi-default                     act/unsup
1003 token-ring-default               act/unsup
1004 fddinet-default                  act/unsup
1005 trnet-default                    act/unsup
! Above, VLANs 2 and 3 do not yet exist. Below, VLAN 2 is added, with name Freds-vlan,
! with two interfaces assigned to VLAN 2.

SW1# configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
SW1(config)# vlan 2
SW1(config-vlan)# name Freds-vlan
SW1(config-vlan)# exit
SW1(config)# interface range fastethernet 0/13 - 14
SW1(config-if)# switchport access vlan 2
SW1(config-if)# switchport mode access
SW1(config-if)# end

! Below, the show running-config command lists the interface subcommands on
! interfaces Fa0/13 and Fa0/14.
SW1# show running-config
! Many lines omitted for brevity
! Early in the output:
vlan 2
 name Freds-vlan
!
! more lines omitted for brevity
interface FastEthernet0/13
 switchport access vlan 2
 switchport mode access
!
interface FastEthernet0/14
 switchport access vlan 2
 switchport mode access
!
```

```
SW1# show vlan brief

VLAN Name                             Status    Ports
---- -------------------------------- --------- -------------------------------
1    default                          active    Fa0/1, Fa0/2, Fa0/3, Fa0/4
                                                Fa0/5, Fa0/6, Fa0/7, Fa0/8
                                                Fa0/9, Fa0/10, Fa0/11, Fa0/12
                                                Fa0/15, Fa0/16, Fa0/17, Fa0/18
                                                Fa0/19, Fa0/20, Fa0/21, Fa0/22
                                                Fa0/23, Fa0/24, Gi0/1, Gi0/2
2    Freds-vlan                       active    Fa0/13, Fa0/14
1002 fddi-default                     act/unsup
1003 token-ring-default               act/unsup
1004 fddinet-default                  act/unsup
1005 trnet-default                    act/unsup

SW1# show vlan id 2
VLAN Name                             Status    Ports
---- -------------------------------- --------- -------------------------------
2   Freds-vlan                        active    Fa0/13, Fa0/14

VLAN Type  SAID       MTU   Parent RingNo BridgeNo Stp  BrdgMode Trans1 Trans2
---- ----- ---------- ----- ------ ------ -------- ---- -------- ------ ------
2    enet  100010     1500  -      -      -        -    -        0      0

Remote SPAN VLAN
----------------
Disabled

Primary Secondary Type              Ports
------- --------- ----------------- ------------------------------------------
```

The example begins with the **show vlan brief** command, confirming the default settings of five nondeletable VLANs, with all interfaces assigned to VLAN 1. (VLAN 1 cannot be deleted, but can be used. VLANs 1002–1005 cannot be deleted and cannot be used as access VLANs today.) In particular, note that this 2960 switch has 24 Fast Ethernet ports (Fa0/1–Fa0/24) and two Gigabit Ethernet ports (Gi0/1 and Gi0/2), all of which are listed as being in VLAN 1 per that first command's output.

Next, the example shows the process of creating VLAN 2 and assigning interfaces Fa0/13 and Fa0/14 to VLAN 2. Note in particular that the example uses the **interface range** command, which causes the **switchport access vlan 2** interface subcommand to be applied to both interfaces in the range, as confirmed in the **show running-config** command output at the end of the example.

After the configuration has been added, to list the new VLAN, the example repeats the **show vlan brief** command. Note that this command lists VLAN 2, name Freds-vlan, and the interfaces assigned to that VLAN (Fa0/13 and Fa0/14). The **show vlan id 2** command that follows then confirms that ports Fa0/13 and Fa0/14 are assigned to VLAN 2.

The example surrounding Figure 11-11 uses six switch ports, all of which need to operate as access ports. That is, each port should not use trunking, but instead should be assigned to a single VLAN, as assigned by the **switchport access vlan** *vlan-id* command. However, as configured in Example 11-1, these interfaces could negotiate to later become trunk ports, because the switch defaults to allow the port to negotiate trunking and decide whether to act as an access interface or as a trunk interface.

For ports that should always act as access ports, add the optional interface subcommand **switchport mode access**. This command tells the switch to only allow the interface to be an access interface. The upcoming section "VLAN Trunking Configuration" discusses more details about the commands that allow a port to negotiate whether it should use trunking.

NOTE The book includes a video that works through a different VLAN configuration example as well. You can find the video on the DVD and on the companion website.

VLAN Configuration Example 2: Shorter VLAN Configuration

Example 11-1 shows several of the optional configuration commands, with a side effect of being a bit longer than is required. Example 11-2 shows a much briefer alternative configuration, picking up the story where Example 11-1 ended and showing the addition of VLAN 3 (as shown in Figure 11-11). Note that SW1 does not know about VLAN 3 at the beginning of this example.

Example 11-2 *Shorter VLAN Configuration Example (VLAN 3)*

```
SW1# configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
SW1(config)# interface range Fastethernet 0/15 - 16
SW1(config-if-range)# switchport access vlan 3
% Access VLAN does not exist. Creating vlan 3
SW1(config-if-range)# ^Z

SW1# show vlan brief

VLAN Name                             Status    Ports
---- -------------------------------- --------- -------------------------------
1    default                          active    Fa0/1, Fa0/2, Fa0/3, Fa0/4
                                                Fa0/5, Fa0/6, Fa0/7, Fa0/8
                                                Fa0/9, Fa0/10, Fa0/11, Fa0/12
                                                Fa0/17, Fa0/18, Fa0/19, Fa0/20
                                                Fa0/21, Fa0/22, Fa0/23, Fa0/24
                                                Gi0/1, Gi0/2
2    Freds-vlan                       active    Fa0/13, Fa0/14
3    VLAN0003                         active    Fa0/15, Fa0/16
1002 fddi-default                     act/unsup
1003 token-ring-default               act/unsup
1004 fddinet-default                  act/unsup
1005 trnet-default                    act/unsup
```

Example 11-2 shows how a switch can dynamically create a VLAN—the equivalent of the **vlan** *vlan-id* global config command—when the **switchport access vlan** interface subcommand refers to a currently unconfigured VLAN. This example begins with SW1 not knowing about VLAN 3. When the **switchport access vlan 3** interface subcommand was used, the switch realized that VLAN 3 did not exist, and as noted in the shaded message in the example, the switch created VLAN 3, using a default name (VLAN0003). No other steps are required to create the VLAN. At the end of the process, VLAN 3 exists in the switch, and interfaces Fa0/15 and Fa0/16 are in VLAN 3, as noted in the shaded part of the **show vlan brief** command output.

VLAN Trunking Protocol

Before showing more configuration examples, you also need to know something about a Cisco protocol and tool called the VLAN Trunking Protocol (VTP). VTP is a Cisco proprietary tool on Cisco switches that advertises each VLAN configured in one switch (with the **vlan** *number* command) so that all the other switches in the campus learn about that VLAN. However, for various reasons, many enterprises choose not to use VTP.

This book does not discuss VTP as an end to itself. However, VTP has some small impact on how every Cisco Catalyst switch works, even if you do not try to use VTP. This brief section introduces enough details of VTP so that you can see these small differences in VTP that cannot be avoided.

This book attempts to ignore VTP as much as is possible. To that end, all examples in this book use switches that have either been set to use VTP transparent mode (with the **vtp mode transparent** global command) or to disable it (with the **vtp mode off** global command). Both options allow the administrator to configure both standard- and extended-range VLANs, and the switch lists the **vlan** commands in the running-config file.

Finally, on a practical note, if you happen to do lab exercises with real switches or with simulators, and you see unusual results with VLANs, check the VTP status with the **show vtp status** command. If your switch uses VTP server or client mode, you will find:

- The server switches can configure VLANs in the standard range only (1–1005).
- The client switches cannot configure VLANs.
- Both servers and clients may be learning new VLANs from other switches, and seeing their VLANs deleted by other switches, because of VTP.
- The **show running-config** command does not list any **vlan** commands.

If possible in lab, switch to VTP transparent mode and ignore VTP for your switch configuration practice until you are ready to focus on how VTP works when studying for the ICND2 exam topics.

NOTE Do not change VTP settings on any switch that also connects to the production network until you know how VTP works and you talk with experienced colleagues. If the switch you configure connects to other switches, which in turn connect to switches used in the production LAN, you could accidentally change the VLAN configuration in other switches with serious impact to the operation of the network. Be careful and never experiment with VTP settings on a switch unless it, and the other switches connected to it, have absolutely no physical links connected to the production LAN.

VLAN Trunking Configuration

Trunking configuration between two Cisco switches can be very simple if you just statically configure trunking. For example, if two Cisco 2960 switches connect to each other, they support only 802.1Q and not ISL. You could literally add one interface subcommand for the switch interface on each side of the link (**switchport mode trunk**), and you would create a VLAN trunk that supported all the VLANs known to each switch.

However, trunking configuration on Cisco switches includes many more options, including several options for dynamically negotiating various trunking settings. The configuration can either predefine different settings or tell the switch to negotiate the settings, as follows:

- **The type of trunking:** IEEE 802.1Q, ISL, or negotiate which one to use
- **The administrative mode:** Whether to always trunk, always not trunk, or negotiate

First, consider the type of trunking. Cisco switches that support ISL and 802.1Q can negotiate which type to use, using the Dynamic Trunking Protocol (DTP). If both switches support both protocols, they use ISL; otherwise, they use the protocol that both support. Today, many Cisco switches do not support the older ISL trunking protocol. Switches that support both types of trunking use the **switchport trunk encapsulation** {**dot1q** | **isl** | **negotiate**} interface subcommand to either configure the type or allow DTP to negotiate the type.

DTP can also negotiate whether the two devices on the link agree to trunk at all, as guided by the local switch port's administrative mode. The administrative mode refers to the configuration setting for whether trunking should be used. Each interface also has an *operational* mode, which refers to what is currently happening on the interface, and might have been chosen by DTP's negotiation with the other device. Cisco switches use the **switchport mode** interface subcommand to define the administrative trunking mode, as listed in Table 11-2.

Key Topic

Table 11-2 Trunking Administrative Mode Options with the **switchport mode** Command

Command Option	Description
access	Always act as an access (nontrunk) port
trunk	Always act as a trunk port
dynamic desirable	Initiates negotiation messages and responds to negotiation messages to dynamically choose whether to start using trunking
dynamic auto	Passively waits to receive trunk negotiation messages, at which point the switch will respond and negotiate whether to use trunking

For example, consider the two switches shown in Figure 11-12. This figure shows an expansion of the network of Figure 11-11, with a trunk to a new switch (SW2) and with parts of VLANs 1 and 3 on ports attached to SW2. The two switches use a Gigabit Ethernet link for the trunk. In this case, the trunk does not dynamically form by default, because both (2960) switches default to an administrative mode of *dynamic auto*, meaning that neither switch initiates the trunk negotiation process. By changing one switch to use *dynamic desirable* mode, which does initiate the negotiation, the switches negotiate to use trunking, specifically 802.1Q because the 2960s support only 802.1Q.

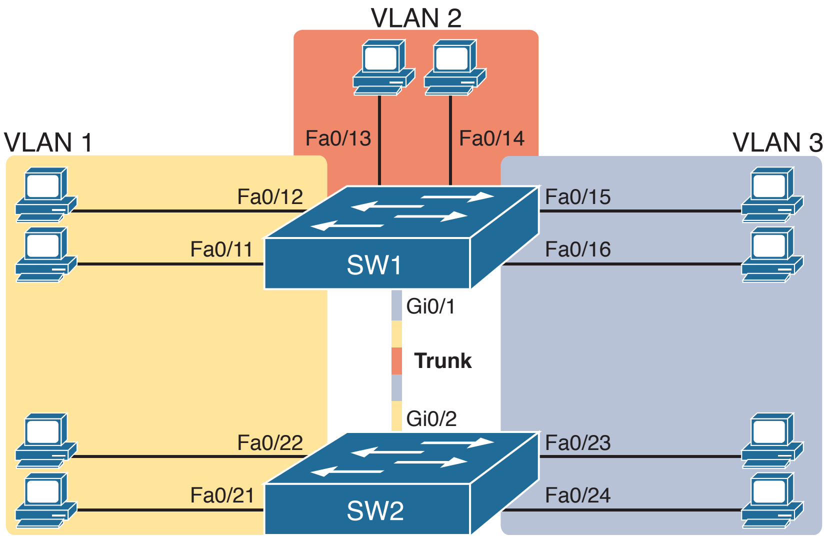


Figure 11-12 *Network with Two Switches and Three VLANs*

Example 11-3 begins by showing the two switches in Figure 11-12 with the default configuration so that the two switches do not trunk.

Example 11-3 *Initial (Default) State: Not Trunking Between SW1 and SW2*

```
SW1# show interfaces gigabit 0/1 switchport
Name: Gi0/1
Switchport: Enabled
Administrative Mode: dynamic auto
Operational Mode: static access
Administrative Trunking Encapsulation: dot1q
Operational Trunking Encapsulation: native
Negotiation of Trunking: On
Access Mode VLAN: 1 (default)
Trunking Native Mode VLAN: 1 (default)
Administrative Native VLAN tagging: enabled
Voice VLAN: none
Access Mode VLAN: 1 (default)
Trunking Native Mode VLAN: 1 (default)
Administrative Native VLAN tagging: enabled
Voice VLAN: none
Administrative private-vlan host-association: none
Administrative private-vlan mapping: none
Administrative private-vlan trunk native VLAN: none
Administrative private-vlan trunk Native VLAN tagging: enabled
Administrative private-vlan trunk encapsulation: dot1q
Administrative private-vlan trunk normal VLANs: none
Administrative private-vlan trunk private VLANs: none
```

```
Operational private-vlan: none
Trunking VLANs Enabled: ALL
Pruning VLANs Enabled: 2-1001
Capture Mode Disabled
Capture VLANs Allowed: ALL

Protected: false
Unknown unicast blocked: disabled
Unknown multicast blocked: disabled
Appliance trust: none

! Note that the next command results in a single empty line of output.
SW1# show interfaces trunk
SW1#
```

First, focus on the highlighted items from the output of the **show interfaces switchport** command at the beginning of Example 11-3. The output lists the default administrative mode setting of dynamic auto. Because SW2 also defaults to dynamic auto, the command lists SW1's operational status as "access," meaning that it is not trunking. ("Dynamic auto" tells both switches to sit there and wait on the other switch to start the negotiations.) The third shaded line points out the only supported type of trunking (802.1Q) on this 2960 switch. (On a switch that supports both ISL and 802.1Q, this value would by default list "negotiate," to mean that the type of encapsulation is negotiated.) Finally, the operational trunking type is listed as "native," which is a reference to the 802.1Q native VLAN.

The end of the example shows the output of the **show interfaces trunk** command, but with no output. This command lists information about all interfaces that currently operationally trunk; that is, it lists interfaces that currently use VLAN trunking. With no interfaces listed, this command also confirms that the link between switches is not trunking.

Next, consider Example 11-4, which shows the new configuration that enables trunking. In this case, SW1 is configured with the **switchport mode dynamic desirable** command, which asks the switch to both negotiate as well as to begin the negotiation process, rather than waiting on the other device. As soon as the command is issued, log messages appear showing that the interface goes down and then back up again, which happens when the interface transitions from access mode to trunk mode.

Example 11-4 *SW1 Changes from Dynamic Auto to Dynamic Desirable*

```
SW1# configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
SW1(config)# interface gigabit 0/1
SW1(config-if)# switchport mode dynamic desirable
SW1(config-if)# ^Z
SW1#
%LINEPROTO-5-UPDOWN: Line protocol on Interface GigabitEthernet0/1, changed state to
  down
%LINEPROTO-5-UPDOWN: Line protocol on Interface GigabitEthernet0/1, changed state to
  up
```

```
SW1# show interfaces gigabit 0/1 switchport
Name: Gi0/1
Switchport: Enabled
Administrative Mode: dynamic desirable
Operational Mode: trunk
Administrative Trunking Encapsulation: dot1q
Operational Trunking Encapsulation: dot1q
Negotiation of Trunking: On
Access Mode VLAN: 1 (default)
Trunking Native Mode VLAN: 1 (default)
! lines omitted for brevity

! The next command formerly listed a single empty line of output; now it lists
! information about the 1 operational trunk.
SW1# show interfaces trunk

Port        Mode         Encapsulation  Status        Native vlan
Gi0/1       desirable    802.1q         trunking      1

Port        Vlans allowed on trunk
Gi0/1       1-4094

Port        Vlans allowed and active in management domain
Gi0/1       1-3

Port        Vlans in spanning tree forwarding state and not pruned
Gi0/1       1-3

SW1# show vlan id 2
VLAN Name                             Status    Ports
---- -------------------------------- --------- -------------------------------
2    Freds-vlan                       active    Fa0/13, Fa0/14, G0/1

VLAN Type  SAID       MTU   Parent RingNo BridgeNo Stp  BrdgMode Trans1 Trans2
---- ----- ---------- ----- ------ ------ -------- ---- -------- ------ ------
2    enet  100010     1500  -      -      -        -    -        0      0

Remote SPAN VLAN
----------------
Disabled

Primary Secondary Type              Ports
------- --------- ----------------- ------------------------------------------
```

To verify whether trunking is working now, the middle of Example 11-4 lists the **show interfaces switchport** command. Note that the command still lists the administrative

11

settings, which denote the configured values along with the operational settings, which list what the switch is currently doing. In this case, SW1 now claims to be in an operational mode of *trunk*, with an operational trunking encapsulation of dot1Q.

The end of the example shows the output of the **show interfaces trunk** command, which now lists G0/1, confirming that G0/1 is now operationally trunking. The next section discusses the meaning of the output of this command.

For the exams, you should be ready to interpret the output of the **show interfaces switchport** command, realize the administrative mode implied by the output, and know whether the link should operationally trunk based on those settings. Table 11-3 lists the combinations of the trunking administrative modes and the expected operational mode (trunk or access) resulting from the configured settings. The table lists the administrative mode used on one end of the link on the left, and the administrative mode on the switch on the other end of the link across the top of the table.

Key Topic

Table 11-3 Expected Trunking Operational Mode Based on the Configured Administrative Modes

Administrative Mode	Access	Dynamic Auto	Trunk	Dynamic Desirable
access	Access	Access	Do Not Use[1]	Access
dynamic auto	Access	Access	Trunk	Trunk
trunk	Do Not Use[1]	Trunk	Trunk	Trunk
dynamic desirable	Access	Trunk	Trunk	Trunk

[1] When two switches configure a mode of "access" on one end and "trunk" on the other, problems occur. Avoid this combination.

Finally, before leaving the discussion of configuring trunks, Cisco recommends disabling trunk negotiation on most ports for better security. The majority of switch ports on most switches will be used to connect to users. As a matter of habit, you can disable DTP negotiations altogether using the **switchport nonegotiate** interface subcommand.

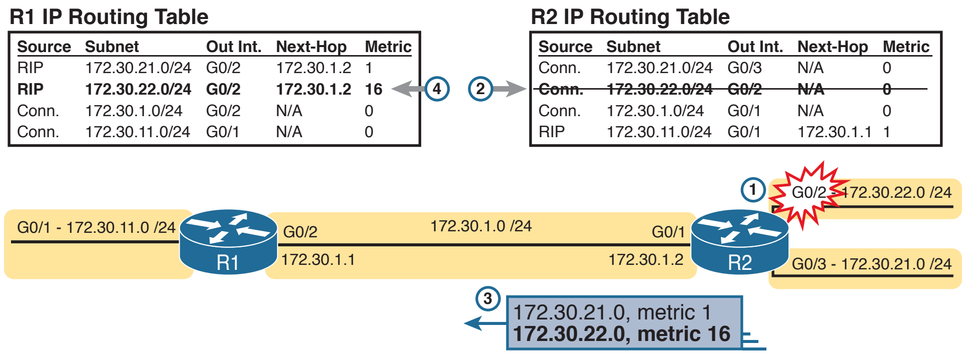
Implementing Interfaces Connected to Phones

This next topic is a strange topic, at least in the context of access links and trunk links. In the world of IP telephony, telephones use Ethernet ports to connect to an Ethernet network so they can use IP to send and receive voice traffic sent via IP packets. To make that work, the switch's Ethernet port acts like an access port—but at the same time, the port acts like a trunk in some ways. This last topic of the chapter works through those main concepts.

Data and Voice VLAN Concepts

Before IP telephony, a PC could sit on the same desk as a phone. The phone happened to use UTP cabling, with that phone connected to some voice device (often called a *voice switch* or a *private branch exchange [PBX]*). The PC, of course, connected using a unshielded twisted-pair (UTP) cable to the usual LAN switch that sat in the wiring closet—sometimes in the same wiring closet as the voice switch. Figure 11-13 shows the idea.

Figure 19-6 shows an example of route poisoning with RIP, with R2's G0/2 interface failing, meaning that R2's route for 172.30.22.0/24 has failed. RIP defines infinity as 16.



R1 IP Routing Table

Source	Subnet	Out Int.	Next-Hop	Metric
RIP	172.30.21.0/24	G0/2	172.30.1.2	1
RIP	**172.30.22.0/24**	**G0/2**	**172.30.1.2**	**16**
Conn.	172.30.1.0/24	G0/2	N/A	0
Conn.	172.30.11.0/24	G0/1	N/A	0

R2 IP Routing Table

Source	Subnet	Out Int.	Next-Hop	Metric
Conn.	172.30.21.0/24	G0/3	N/A	0
~~Conn.~~	~~172.30.22.0/24~~	~~G0/2~~	~~N/A~~	~~0~~
Conn.	172.30.1.0/24	G0/1	N/A	0
RIP	172.30.11.0/24	G0/1	172.30.1.1	1

Figure 19-6 *Route Poisoning*

Figure 19-6 shows the following process, again following the numbers in the figure:

1. R2's G0/2 interface fails.
2. R2 removes its connected route for 172.30.22.0/24 from its routing table.
3. R2 advertises 172.30.22.0 with an infinite metric (which for RIP is 16).
4. R1 realizes that the route to 172.30.22.0/24 no longer works. Depending on conditions not discussed here, R1 either removes the route from its routing table or marks the route as unusable (with an infinite metric) for a few minutes before removing the route.

By the end of this process, Router R1 knows for sure that its old route for subnet 172.30.22.0/24 has failed, which helps R1 avoid introducing looping IP routes.

Note that all routing protocols have mechanisms to use to mark routes as expired in some way, in some cases using an infinite metric value similar to RIP. RIP uses 16 per the RIP protocol definition; as a result, a route with hop count 15 is the longest valid route that can be used in a RIP network, because advertising a route with hop count 16 would be considered a poison route.

Summarizing RIPv2 Features

This final section briefly mentions a few more features of RIPv2, and collects those features into a table for easier review and study.

Of course, RIPv2 adds features beyond RIPv1. For instance, RIPv2 supports authentication, which is a feature by which routers can use a password-like mechanism to make sure they exchange routes only with authentic other routers. RIPv2 also supports manual route summarization, which allows an engineer to plan and reduce the size of routing tables. (The DVD Appendix O, "Route Summarization," copied from a previous edition of this book, provides more detail if you are interested.) However, this book does not get into details about these features beyond this brief mention.

For another difference, RIPv2 sends its update message—the message that lists routing information—to the 224.0.0.9 multicast IP address. RIPv1 used the local subnet broadcast address of 255.255.255.255. Using the multicast address is more efficient and causes less impact to other hosts.

Finally, RIPv2 adds support for variable-length subnet masks (VLSM). Chapter 22, "Variable Length Subnet Masks," goes into detail about VLSM. To review, VLSM means that inside one classful network (one Class A, B, or C network), more than one subnet mask is used. For instance, the network in Figure 19-7 uses VLSM because all the subnets are from Class A network 10.0.0.0, but some subnets use a /24 mask whereas others use a /30 mask.

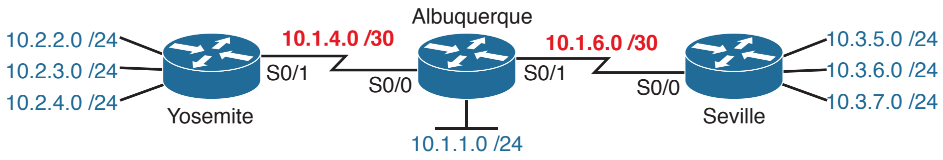


Figure 19-7 *An Example of VLSM*

Table 19-2 lists the features comparing RIPv1 and RIPv2. However, note that the list of features in the table is more about emphasizing the features of RIPv2 than stressing the differences between the two versions.

Key Topic

Table 19-2 Key Features of RIPv1 and RIPv2

Feature	RIPv1	RIPv2
Hop-count metric	Yes	Yes
Sets 15 as the largest metric for a working route	Yes	Yes
Sends full routing updates	Yes	Yes
Uses split horizon	Yes	Yes
Uses route poisoning, with metric 16 to mean "infinite"	Yes	Yes
Sends mask in routing update, thereby supporting VLSM	No	Yes
Supports manual route summarization	No	Yes
Sends updates to 224.0.0.9 multicast address	No	Yes
Supports authentication	No	Yes

Core RIPv2 Configuration and Verification

RIPv2 requires three basic configuration commands, with just a couple of **show** commands to check RIPv2 status. This second of four major sections of this chapter focuses on that core configuration and verification.

Configuring Core RIPv2 Features

RIPv2 configuration is simple compared to the concepts related to routing protocols. The configuration process uses three required commands, with only one command, the **network** command, requiring any real thought. You should also know the more popular **show** commands for helping you analyze and troubleshoot routing protocols.

The RIPv2 configuration process takes only the following three required steps, with the possibility that the third step might need to be repeated several times on the same router:

Config Checklist

Step 1. Use the **router rip** command in global configuration mode to move into RIP configuration mode.

Step 2. Use the **version 2** command in RIP configuration mode to tell the router to use RIP Version 2 exclusively.

Step 3. Use one or more **network** *net-number* commands in RIP configuration mode to enable RIP on the correct interfaces.

Understanding the RIP network Command

To configure RIPv2, always start with those first two commands in the configuration checklist, and then think hard about the third step, the **network** command. The RIP **network** indirectly identifies the interfaces on which RIP is then enabled. The command has one parameter: some classful IP network number. IOS then compares each interface IP address of each interface on the local router with the IP network in the **network** command. IOS enables RIP on each interface whose IP address is in that same classful network.

For example, in Figure 19-8, the configuration on the left uses two **network** commands. The first **network** command happens to match one interface IP address of the four interfaces on the right, because one of the interfaces is in classful network 10.0.0.0. The second command matches two interfaces, because both are in classful network 172.16.0.0. Neither of the two **network** commands match the fourth interface, which is in classful network 192.168.1.0.

Key Topic

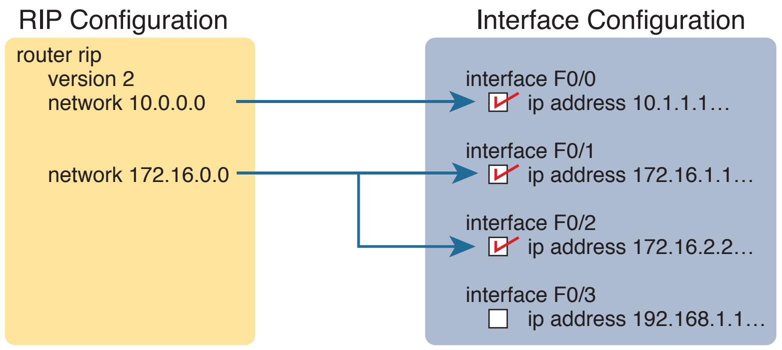


Figure 19-8 *RIP Network Command Enabling RIP Per-Interface Logic*

So, what does RIPv2 do on an interface once enabled? Well, RIP takes three separate actions once enabled on an interface. So rather than think of enabling RIP on an interface as one idea, break it into these three actions, which will help when you think about some later configuration and troubleshooting topics. The following are the three actions:

Key Topic

- The router sends routing updates out the interface.
- The router listens for and processes incoming updates on that same interface.
- The router advertises about the subnet connected to the interface.

Note that with the **version 2** command configured, the updates sent and received per this list are RIP version 2 updates.

RIP Configuration Example, with Many IP Networks

Keeping these facts in mind, now consider how to configure RIP on a single router. Examine Figure 19-9 for a moment and try to apply the first three configuration steps to this router and anticipate the configuration required on the router to enable RIP on all interfaces.

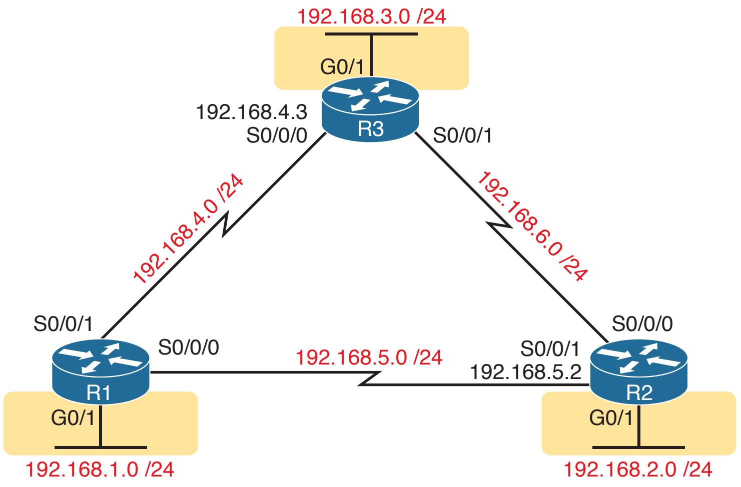


Figure 19-9 *RIPv2 Configuration: Three Routers, Each Connected to Three Networks*

Take a close look at the IP subnets listed in the figure. All links use an entire Class C network. I chose to use different Class C networks on each link on purpose so that each router connects to multiple classful networks. For instance, R2 will need **network** commands for networks 192.68.2.0, 192.168.5.0, and 192.168.6.0. Example 19-1 shows the configuration for all three routers.

Example 19-1 *R1, R2, and R3 RIPv2 Configuration, for Figure 19-9*

```
! Router R1 configuration
router rip
 version 2
 network 192.168.1.0
 network 192.168.4.0
 network 192.168.5.0
```

```
! Router R2 configuration
router rip
 version 2
 network 192.168.2.0
 network 192.168.5.0
 network 192.168.6.0
```

```
! Router R3 configuration
router rip
 version 2
 network 192.168.3.0
 network 192.168.4.0
 network 192.168.6.0
```

First, focus on meeting the primary goals. All three routers have the **router rip** and **version 2** commands, which together enable RIPv2, but without enabling RIPv2 on any interfaces.

Next, focus on the three **network** commands on each router. Each router has three **network** commands in this example because each router connects to three different classful networks. For example, R1 lists IP network 192.168.1.0, 192.168.4.0, and 192.168.5.0 in its three **network** commands, because according to the figure, R1 connects directly to those three IP networks. Those three commands enable RIPv2 on R1's G0/1, S0/0/0, and S0/0/1 interfaces. The **network** commands on the other two routers enable RIPv2 on their interfaces, respectively.

This particular example configuration gives us a good backdrop to discuss a common question about RIPv2 configuration and **network** commands. First, no one router has **network** commands for all six classful IP network numbers. The **network** command does not predefine all classful networks in the entire topology. Instead, it triggers local logic on that one router, matching the **network** commands against the interface IP addresses on that one router, as shown earlier in Figure 19-8.

Finally, on a complete side note about the RIP **network** command: IOS will actually accept a parameter besides a classful network number. IOS will not even issue an error message. However, IOS, knowing that the parameter must be a classful network number, interprets the IP address and changes the number to the matching network number. For example, if you were to type **network 10.1.2.3** in RIP configuration mode, IOS would accept the command, with no error message, and change what you typed so that the configuration has a **network 10.0.0.0** command. Your original **network 10.1.2.3** command would disappear.

RIP Configuration Example, with One IP Network

Figure 19-9, used in the first RIP configuration example, purposefully used many IP networks so that the configuration required several RIPv2 **network** commands. However, often a design will use subnets of one classful network, as shown in Figure 19-10. In this case, all six subnets are subnets of Class A network 10.0.0.0. Note that Figures 19-9 and 19-10 are identical other than the IPv4 subnets used.

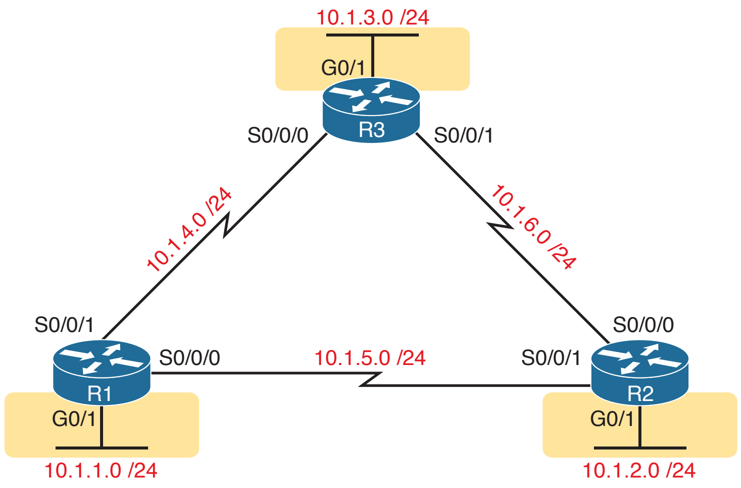


Figure 19-10 *Three Routers, Each Connected to Subnets of Class A Network 10.0.0.0*

To enable RIPv2 on all interfaces, each router needs only one **network** command: the **network 10.0.0.0** command. That one command on a router matches all three interfaces. Example 19-2 shows the identical configuration used by all three routers.

Example 19-2 *The Identical RIPv2 Configuration Used for R1, R2, R3 for Figure 19-10*

```
router rip
 version 2
 network 10.0.0.0
```

RIPv2 Verification

IOS includes three primary **show** commands that are helpful to confirm how well RIPv2 is working. Table 19-3 lists the commands and their main purpose.

Key Topic

Table 19-3 RIP Operational Commands

Command	Purpose
show ip route [**rip**]	Routes: This command lists IPv4 routes as learned by RIP. The **show ip route** command lists all IPv4 routes, and the **show ip route rip** command lists RIP-learned routes only.
show ip protocols	Configuration: This command lists information about the RIP configuration, plus the IP addresses of neighboring RIP routers from which the local router has learned routes.
show ip rip database	Best routes: This command lists the prefix/length of all best routes known to RIP on this router, including routes learned from neighbors and connected routes for interfaces on which RIP has been enabled.

Examining RIP Routes in the IP Routing Table

To begin, consider Router R1's routing table, based on Figure 19-9 and the configuration in Example 19-1. That is the sample with six different Class C networks in the three-router design. Example 19-3 shows the full IP routing table, as well as just the RIP-learned routes.

Example 19-3 *The* **show ip route** *Command*

```
R1# show ip route
Codes: L - local, C - connected, S - static, R - RIP, M - mobile, B - BGP
       D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
       E1 - OSPF external type 1, E2 - OSPF external type 2
       i - IS-IS, su - IS-IS summary, L1 - IS-IS level-1, L2 - IS-IS level-2
       ia - IS-IS inter area, * - candidate default, U - per-user static route
       o - ODR, P - periodic downloaded static route, H - NHRP, l - LISP
       a - application route
       + - replicated route, % - next hop override

Gateway of last resort is not set

      192.168.1.0/24 is variably subnetted, 2 subnets, 2 masks
C        192.168.1.0/24 is directly connected, GigabitEthernet0/1
L        192.168.1.1/32 is directly connected, GigabitEthernet0/1
```

```
R       192.168.2.0/24 [120/1] via 192.168.5.2, 00:00:21, Serial0/0/0
R       192.168.3.0/24 [120/1] via 192.168.4.3, 00:00:05, Serial0/0/1
        192.168.4.0/24 is variably subnetted, 2 subnets, 2 masks
C          192.168.4.0/24 is directly connected, Serial0/0/1
L          192.168.4.1/32 is directly connected, Serial0/0/1
        192.168.5.0/24 is variably subnetted, 2 subnets, 2 masks
C          192.168.5.0/24 is directly connected, Serial0/0/0
L          192.168.5.1/32 is directly connected, Serial0/0/0
R       192.168.6.0/24 [120/1] via 192.168.5.2, 00:00:21, Serial0/0/0
                       [120/1] via 192.168.4.3, 00:00:05, Serial0/0/1
R1# show ip route rip
! The same lines of legend show up here - removed for brevity

R       192.168.2.0/24 [120/1] via 192.168.5.2, 00:00:21, Serial0/0/0
R       192.168.3.0/24 [120/1] via 192.168.4.3, 00:00:05, Serial0/0/1
R       192.168.6.0/24 [120/1] via 192.168.5.2, 00:00:21, Serial0/0/0
                       [120/1] via 192.168.4.3, 00:00:05, Serial0/0/1
```

First, scan around all the detail in the **show ip route** command. Notice the legend at the top—about 10 lines of output—which are the same for every **show ip route** command. The legend lists the routing codes, which are short codes that identify the source from which a route is learned. In this case, Router R1 IPv4 routes with three codes—C, L, and R—meaning connected, local, and RIP.

Scan farther down in the output to see the individual routes. The routes list the subnet and then mask in prefix format, followed by other details. Ignoring the detail for another moment, notice the three highlighted RIP-learned routes, both in the output of the **show ip route** command and in the **show ip route rip** command at the end of the example. The highlighted lines show the same routes, but the **show ip route rip** command lists only the RIP routes, and not the connected and local routes.

Each line in the output of these commands reveals many details about a route. Using R1's route to 192.168.2.0/24 as an example, the details are as follows:

Key Topic

- The network number and mask are listed, 192.168.2.0 and /24 in this case. (In some cases, the mask is on a heading line just above the route.)
- The next-hop router's IP address is 192.168.5.2 in this case.
- The outgoing interface is Serial0/0/0 in this case.
- The update RIP timer that measures how long it has been since R1 has last heard about this route in a periodic RIP update is 21 seconds ago in this case.
- The RIP metric for this route (1 in this case), is listed as the second number in the square brackets. For example, between R1 and subnet 192.168.2.0/24, one other router (R2) exists, so it is a one-hop route.
- The administrative distance of the route is 120 in this case; the first number in brackets.

Take the time now to review the other two RIP routes, noting the values for these various items in those routes.

To better understand RIP metrics, think for a moment what would happen in this three-router topology of Figure 19-9 if R1's S0/0/0 interface failed. R1's one-hop route to 192.168.2.0/24 uses that outgoing interface, but if it were to fail, R1 would converge to use the two-hop route that runs through R3 next. Example 19-4 shows the RIP routes on R1 again after that failure.

Example 19-4 *The* **show ip route** *Command with New Metric 2 for Subnet 192.168.2.0*

```
R1# show ip route rip
! The same lines of legend show up here - removed for brevity

R      192.168.2.0/24 [120/2] via 192.168.4.3, 00:00:01, Serial0/0/1
R      192.168.3.0/24 [120/1] via 192.168.4.3, 00:00:01, Serial0/0/1
R      192.168.6.0/24 [120/1] via 192.168.4.3, 00:00:01, Serial0/0/1
```

NOTE In lab, all you have to do to re-create the link failure in this example is to issue a **shutdown** command under R1's S0/0/0 interface.

Examine the highlighted route in detail, and compare it to R1's route for 192.168.2.0 from the previous example. In this case, the network, mask, and administrative distance remain the same. However, the metric is now 2, because this route goes through R3 and then R2, for two hops. It also lists forwarding directions of going through 192.168.4.3 (R3's S0/0/0 IP address) and going out R1's S0/0/1 interface.

Comparing Routing Sources with Administrative Distance

As you just examined, when an internetwork has redundant links and uses a single routing protocol, each router may learn multiple routes to reach a particular subnet. That one routing protocol then uses a metric to choose the best route, and the router adds that route to its routing table. For instance, R1 uses the metric 1 route for 192.168.2.0/24 when all links were working, and then the metric 2 route for 192.168.2.0/24 when a link in that better route failed.

However, some enterprises use multiple IP routing protocols. One router might learn of multiple routes to a particular subnet using different routing protocols. In these cases, the metric does not help the router choose which route is best, because each routing protocol uses a metric unique to that routing protocol. For example, RIP uses the hop count as the metric, but EIGRP uses a math formula with bandwidth and delay as inputs. A route with RIP metric 1 might need to be compared to an EIGRP route, to the same subnet, but with metric 4,132,768. (Yes, EIGRP metrics tend to be large numbers.) Because the numbers have different meanings, there is no real way to compare the metrics.

The router still needs to choose the best route even between routes learned by different routing protocols, or between routing protocols and static routes. IOS solves this problem by assigning a numeric value to each routing protocol. IOS then chooses the route whose routing protocol has the lower number. This number is called the administrative distance (AD). For example, EIGRP defaults to use an AD of 90, and RIP defaults to use the value of 120, as shown in the routes in Example 19-3 and 19-4. Given the lower-is-better logic used with administrative distance, an EIGRP route to a subnet would be chosen instead of a competing RIP route.

Table 19-4 lists the AD values for the most common sources of routing information.

Key Topic

Table 19-4 IOS Defaults for Administrative Distance

Route Source	Administrative Distance
Connected routes	0
Static routes	1
EIGRP	90
OSPF	110
RIP (v1 and v2)	120
DHCP default route	254
Unknown or unbelievable	255

NOTE You might recall a brief mention of administrative distance in Chapter 18, "Configuring IPv4 Addresses and Static Routes." That chapter explained how to configure a floating static route that was used only when the routing protocol did not learn a route for a given subnet, using administrative distance as the key mechanism.

Revealing RIP Configuration with the show ip protocols Command

The **show ip route** command shows the end result of RIP's work, but the **show ip protocols** command tells us something about how RIP works and about the RIP configuration. Example 19-5 lists the output of this command, taken from Router R1 in Figure 19-9, again based on the configuration in Example 19-1.

Example 19-5 *The* **show ip protocols** *Command Based on Example 19-1 Configuration*

```
R1# show ip protocols
Routing Protocol is "rip"
  Outgoing update filter list for all interfaces is not set
  Incoming update filter list for all interfaces is not set
  Sending updates every 30 seconds, next due in 23 seconds
  Invalid after 180 seconds, hold down 180, flushed after 240
  Redistributing: rip
  Default version control: send version 2, receive version 2
    Interface             Send  Recv  Triggered RIP  Key-chain
    GigabitEthernet0/1    2     2
    Serial0/0/0           2     2
    Serial0/0/1           2     2
  Automatic network summarization is in effect
  Maximum path: 4
  Routing for Networks:
    192.168.1.0
    192.168.4.0
    192.168.5.0
Routing Information Sources:
    Gateway         Distance      Last Update
    192.168.4.3          120      00:00:18
    192.168.5.2          120      00:00:05
  Distance: (default is 120)
```

The example highlights the configuration information that can be gleaned from the output, at least for configuration commands mentioned in this chapter, as follows:

- The **version 2** RIP subcommand configured R1 to send version 2 updates only and to process received version 2 updates only as well.
- Automatic summarization (per the default **auto-summary** command) is enabled.
- The **maximum-paths** command is set to 4 (also the default).
- The "Routing for Networks" section re-lists the configuration of **network** commands, listing the network numbers in those commands. In this case, it implies three RIP subcommands: **network 192.168.1.0**, **network 192.168.4.0**, and **network 192.168.5.0**.

The output also lists RIP status information. For instance, look at the bottom of the example, to the "Routing Information Sources" section (not highlighted). It lists two gateways (that is, routers) by IP address. This list is the list of IP addresses of neighboring routers from which R1 has heard RIP updates. (If you look back to Figure 19-9, you will see these two IP addresses as addresses on R3 and R2, respectively.) The last update timer lists the time since R1 last heard from the neighbor; remember, RIP relies on the periodic receipt of RIP updates to know that the neighbor still exists. In short, this list shows that R1 is learning RIP routes from these two routers.

Examining the Best RIP Routes Using RIP Database

One more command can reveal some important details about RIP operation on a router: the **show ip rip database** command. This command lists the prefix/length of each subnet known to the local router's RIP process. This command lists the local router's best route to each known subnet. In particular, this command lists

- Routes for subnets learned from other RIP routers
- Routes for connected subnets for which RIP is enabled on interfaces due to the RIP **network** command(s)

The fact that the **show ip rip database** command lists both learned routes and connected routes for RIP-enabled interfaces makes this command unique. In comparison, Examples 19-3 and 19-4 show samples of the **show ip route** command from R1 in Figure 19-10, listing RIP-learned routes. However, you cannot tell for sure on which interfaces RIP has been enabled based on this output. Example 19-5 shows how the **show ip protocols** command identifies the interfaces on which RIP is enabled, with three interfaces on that same Router R1. However, this command does not identify any RIP-learned routes.

As shown in Example 19-6, the **show ip rip database** command lists both connected and RIP-learned routes. The sample is again taken from Router R1 in Figure 19-10, with all interfaces working. The output identifies the same three RIP-learned routes listed in Example 19-4, with the hop-count metrics in brackets and the next-hop router IP addresses listed (R2, address 10.1.5.2, and R3, 10.1.4.3). The output also lists the connected subnets, identifying the interfaces on which RIP has been enabled.

Example 19-6 *The* **show ip rip database** *Command on Router R1 (Figure 19-10)*

```
R1# show ip rip database
10.0.0.0/8    auto-summary
10.1.1.0/24    directly connected, GigabitEthernet0/1
10.1.2.0/24
    [1] via 10.1.5.2, 00:00:00, Serial0/0/0
10.1.3.0/24
    [1] via 10.1.4.3, 00:00:08, Serial0/0/1
10.1.4.0/24    directly connected, Serial0/0/1
10.1.5.0/24    directly connected, Serial0/0/0
10.1.6.0/24
    [1] via 10.1.5.2, 00:00:00, Serial0/0/0
    [1] via 10.1.4.3, 00:00:08, Serial0/0/1
```

Optional RIPv2 Configuration and Verification

This next major section, the third of four major sections in this chapter, introduces a few optional RIPv2 features. The features are passive interfaces, maximum (routing) paths, automatic route summarization, and discontiguous networks.

Controlling RIP Updates with the passive-interface Command

In some cases, you want to enable RIP on an interface so that you can advertise about the connected subnet, but you do not need to advertise routes on that interface. This is typically true for any router LAN interface for which that router interface is the only interface connected to the LAN. No other routers connect to the LAN, so the router does not need to send updates onto the LAN.

The RIPv2 **passive-interface** command can be used to stop all RIPv2 updates from being sent out the interface that is matched by a **network** command. By making an interface passive to RIP, the RIP process no longer sends RIP updates out that interface. RIP will still process any received updates and will still advertise about the connected subnet.

IOS gives us two configuration methods to make interfaces passive. The first is the most obvious: use the **passive-interface** *type number* RIP subcommand for each interface that you want to make passive to RIP.

Example 19-7 shows a revised version of the Router R1 configuration first shown in Example 19-2. In that example, all three routers (from Figure 19-10) connect to subnets of network 10.0.0.0. All three routers also have a G0/1 LAN interface that connects to a LAN that has no other routers connected to it. Example 19-7 shows the original configuration from Example 19-2, with the addition of the **passive-interface** command to make the LAN interface passive.

Example 19-7 *Directing RIPv2 to Not Send Advertisements with* **passive-interface**

```
router rip
 version 2
 network 10.0.0.0
 passive-interface G0/1
```

CHAPTER 25

Basic IPv4 Access Control Lists

This chapter covers the following exam topics:

4.0 Infrastructure Services

4.6 Configure, verify, and troubleshoot IPv4 standard numbered and named access list for routed interfaces

Almost every other topic in the scope of CCENT and CCNA R&S focuses on achieving a core goal of any TCP/IP network: delivering IPv4 packets from the source host to the destination host. This chapter, along with the next chapter, focuses instead on preventing some of those packets from being allowed to reach their destinations, by using IPv4 access control lists (ACL).

IPv4 ACLs have many uses, but the CCENT and CCNA R&S certifications focus on their most commonly known use: as packet filters. You want hosts in one subnet to be able to communicate throughout your corporate network, but maybe there is a pocket of servers with sensitive data that must be protected. Maybe government privacy rules require you to further secure and protect access, not just with usernames and login, but even to protect the ability to deliver a packet to the protected host or server. IP ACLs provide a useful solution to achieve those goals.

IPv4 ACLs give network engineers the ability to program a filter into a router. Each router, on each interface, for both the inbound and outbound direction, can enable a different ACL with different rules. Each ACL's rules tell the router which packets to discard, and which to allow through.

This chapter discusses the basics of IPv4 ACLs, and in particular, one type of IP ACL: standard numbered IP ACLs. Chapter 26, "Advanced IPv4 Access Control Lists," completes the discussion by describing other types of IP ACLs.

"Do I Know This Already?" Quiz

Take the quiz (either here, or use the PCPT software) if you want to use the score to help you decide how much time to spend on this chapter. The answers are at the bottom of the page following the quiz, and the explanations are in DVD Appendix C and in the PCPT software.

Table 25-1 "Do I Know This Already?" Foundation Topics Section-to-Question Mapping

Foundation Topics Section	Questions
IP Access Control List Basics	1
Standard Numbered IPv4 ACLs	2–5
Practice Applying Standard IP ACLs	6

1. Barney is a host with IP address 10.1.1.1 in subnet 10.1.1.0/24. Which of the following are things that a standard IP ACL could be configured to do? (Choose two answers.)
   a. Match the exact source IP address.
   b. Match IP addresses 10.1.1.1 through 10.1.1.4 with one **access-list** command without matching other IP addresses.
   c. Match all IP addresses in Barney's subnet with one **access-list** command without matching other IP addresses.
   d. Match only the packet's destination IP address.
2. Which of the following answers list a valid number that can be used with standard numbered IP ACLs? (Choose two answers.)
   a. 1987
   b. 2187
   c. 187
   d. 87
3. Which of the following wildcard masks is most useful for matching all IP packets in subnet 10.1.128.0, mask 255.255.255.0?
   a. 0.0.0.0
   b. 0.0.0.31
   c. 0.0.0.240
   d. 0.0.0.255
   e. 0.0.15.0
   f. 0.0.248.255
4. Which of the following wildcard masks is most useful for matching all IP packets in subnet 10.1.128.0, mask 255.255.240.0?
   a. 0.0.0.0
   b. 0.0.0.31
   c. 0.0.0.240
   d. 0.0.0.255
   e. 0.0.15.255
   f. 0.0.248.255

5. ACL 1 has three statements, in the following order, with address and wildcard mask values as follows: 1.0.0.0 0.255.255.255, 1.1.0.0 0.0.255.255, and 1.1.1.0 0.0.0.255. If a router tried to match a packet sourced from IP address 1.1.1.1 using this ACL, which ACL statement does a router consider the packet to have matched?

   a. First
   b. Second
   c. Third
   d. Implied deny at the end of the ACL

6. Which of the following **access-list** commands matches all packets sent from hosts in subnet 172.16.4.0/23?

   a. **access-list 1 permit 172.16.0.5 0.0.255.0**
   b. **access-list 1 permit 172.16.4.0 0.0.1.255**
   c. **access-list 1 permit 172.16.5.0**
   d. **access-list 1 permit 172.16.5.0 0.0.0.127**

Foundation Topics

IPv4 Access Control List Basics

IPv4 access control lists (IP ACL) give network engineers a way to identify different types of packets. To do so, the ACL configuration lists values that the router can see in the IP, TCP, UDP, and other headers. For example, an ACL can match packets whose source IP address is 1.1.1.1, or packets whose destination IP address is some address in subnet 10.1.1.0/24, or packets with a destination port of TCP port 23 (Telnet).

IPv4 ACLs perform many functions in Cisco routers, with the most common use as a packet filter. Engineers can enable ACLs on a router so that the ACL sits in the forwarding path of packets as they pass through the router. After it is enabled, the router considers whether each IP packet will either be discarded or allowed to continue as if the ACL did not exist.

However, ACLs can be used for many other IOS features as well. As an example, ACLs can be used to match packets for applying quality of service (QoS) features. QoS allows a router to give some packets better service, and other packets worse service. For example, packets that hold digitized voice need to have very low delay, so ACLs can match voice packets, with QoS logic in turn forwarding voice packets more quickly than data packets.

This first section introduces IP ACLs as used for packet filtering, focusing on these aspects of ACLs: the locations and direction in which to enable ACLs, matching packets by examining headers, and taking action after a packet has been matched.

ACL Location and Direction

Cisco routers can apply ACL logic to packets at the point at which the IP packets enter an interface, or the point at which they exit an interface. In other words, the ACL becomes associated with an interface and for a direction of packet flow (either in or out). That is, the ACL can be applied inbound to the router, before the router makes its forwarding (routing) decision, or outbound, after the router makes its forwarding decision and has determined the exit interface to use.

The arrows in Figure 25-1 show the locations at which you could filter packets flowing left to right in the topology. For example, imagine that you wanted to allow packets sent by host A to server S1, but to discard packets sent by host B to server S1. Each arrowed line represents a location and direction at which a router could apply an ACL, filtering the packets sent by host B.

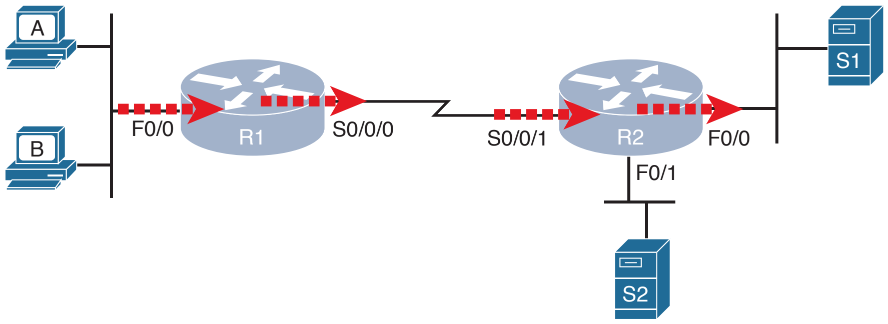


Figure 25-1 *Locations to Filter Packets from Hosts A and B Going Toward Server S1*

The four arrowed lines in the figure point out the location and direction for the router interfaces used to forward the packet from host B to server S1. In this particular example, those interfaces and direction are inbound on R1's F0/0 interface, outbound on R1's S0/0/0 interface, inbound on R2's S0/0/1 interface, and outbound on R2's F0/0 interface. If, for example, you enabled an ACL on R2's F0/1 interface, in either direction, that ACL could not possibly filter the packet sent from host B to server S1, because R2's F0/1 interface is not part of the route from B to S1.

Key Topic

In short, to filter a packet, you must enable an ACL on an interface that processes the packet, in the same direction the packet flows through that interface.

When enabled, the router then processes every inbound or outbound IP packet using that ACL. For example, if enabled on R1 for packets inbound on interface F0/0, R1 would compare every inbound IP packet on F0/0 to the ACL to decide that packet's fate: to continue unchanged, or to be discarded.

Matching Packets

When you think about the location and direction for an ACL, you must already be thinking about what packets you plan to filter (discard), and which ones you want to allow through. To tell the router those same ideas, you must configure the router with an IP ACL that matches packets. *Matching packets* refers to how to configure the ACL commands to look at each packet, listing how to identify which packets should be discarded, and which should be allowed through.

Answers to the "Do I Know This Already?" quiz:

1 A, C **2** A, D **3** D **4** E **5** A **6** B

Each IP ACL consists of one or more configuration commands, with each command listing details about values to look for inside a packet's headers. Generally, an ACL command uses logic like "look for these values in the packet header, and if found, discard the packet." (The action could instead be to allow the packet, rather than discard.) Specifically, the ACL looks for header fields you should already know well, including the source and destination IP addresses, plus TCP and UDP port numbers.

For example, consider an example with Figure 25-2, in which you want to allow packets from host A to server S1, but to discard packets from host B going to that same server. The hosts all now have IP addresses, and the figure shows pseudocode for an ACL on R2. Figure 25-2 also shows the chosen location to enable the ACL: inbound on R2's S0/0/1 interface.

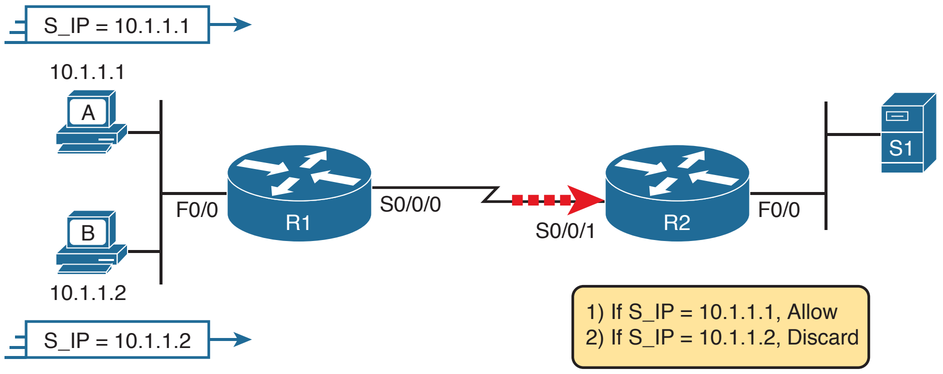


Figure 25-2 *Pseudocode to Demonstrate ACL Command-Matching Logic*

Figure 25-2 shows a two-line ACL in a rectangle at the bottom, with simple matching logic: both statements just look to match the source IP address in the packet. When enabled, R2 looks at every inbound IP packet on that interface and compares each packet to those two ACL commands. Packets sent by host A (source IP address 10.1.1.1) are allowed through, and those sourced by host B (source IP address 10.1.1.2) are discarded.

Taking Action When a Match Occurs

When using IP ACLs to filter packets, only one of two actions can be chosen. The configuration commands use keywords **deny** and **permit**, and they mean (respectively) to discard the packet or to allow it to keep going as if the ACL did not exist.

This book focuses on using ACLs to filter packets, but IOS uses ACLs for many more features. Those features typically use the same matching logic. However, in other cases, the **deny** or **permit** keywords imply some other action. For example, Chapter 27, "Network Address Translation," uses ACLs to match packets, but matching with a **permit** keyword tells the router to apply NAT functions that translate the IP addresses.

Types of IP ACLs

Cisco IOS has supported IP ACLs since the early days of Cisco routers. Beginning with the original standard numbered IP ACLs in the early days of IOS, which could enable the logic shown earlier around Figure 25-2, Cisco has added many ACL features, including the following:

- Standard numbered ACLs (1–99)
- Extended numbered ACLs (100–199)
- Additional ACL numbers (1300–1999 standard, 2000–2699 extended)
- Named ACLs
- Improved editing with sequence numbers

This chapter focuses solely on standard numbered IP ACLs, while the next chapter discusses the other three primary categories of IP ACLs. Briefly, IP ACLs will be either numbered or named in that the configuration identifies the ACL either using a number or a name. ACLs will also be either standard or extended, with extended ACLs having much more robust abilities in matching packets. Figure 25-3 summarizes the big ideas related to categories of IP ACLs.

Key Topic

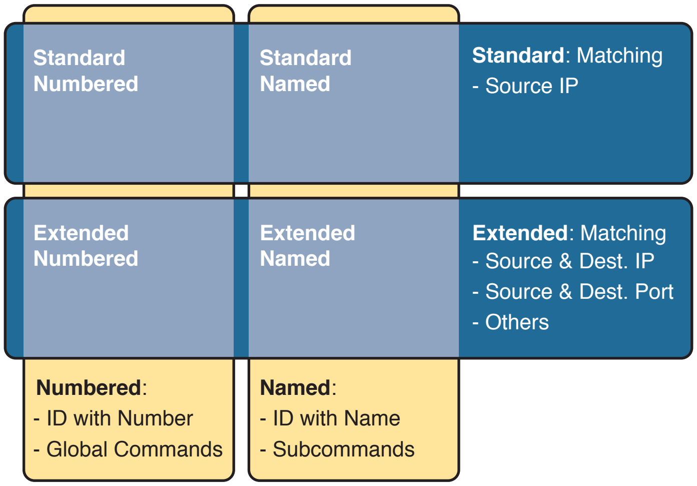


Figure 25-3 *Comparisons of IP ACL Types*

Standard Numbered IPv4 ACLs

The title of this section serves as a great introduction, if you can decode what Cisco means by each specific word. This section is about a type of Cisco filter (ACL) that matches only the source IP address of the packet (*standard*), is configured to identify the ACL using numbers rather than names (*numbered*), and looks at IPv4 packets.

This section examines the particulars of standard numbered IP ACLs. First, it examines the idea that one ACL is a list, and what logic that list uses. Following that, the text closely looks at how to match the source IP address field in the packet header, including the syntax of the commands. This section ends with a complete look at the configuration and verification commands to implement standard ACLs.

List Logic with IP ACLs

A single ACL is both a single entity and, at the same time, a list of one or more configuration commands. As a single entity, the configuration enables the entire ACL on an interface, in a specific direction, as shown earlier in Figure 25-1. As a list of commands, each command has different matching logic that the router must apply to each packet when filtering using that ACL.

When doing ACL processing, the router processes the packet, compared to the ACL, as follows:

Key Topic

ACLs use first-match logic. Once a packet matches one line in the ACL, the router takes the action listed in that line of the ACL, and stops looking further in the ACL.

To see exactly what that means, consider the example built around Figure 25-4. The figure shows an example ACL 1 with three lines of pseudocode. This example applies ACL 1 on R2's S0/0/1 interface, inbound (the same location as in earlier Figure 25-2).

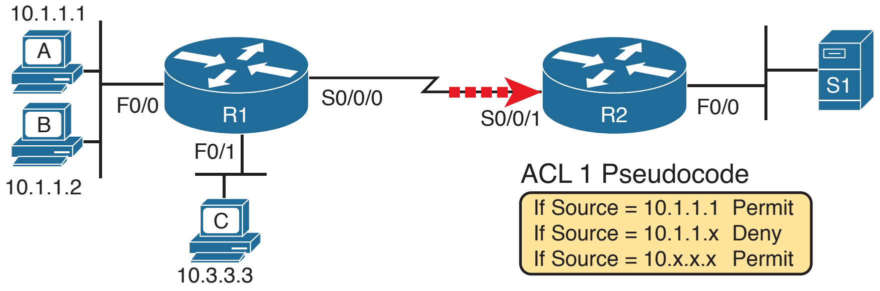

10.1.1.1
A
B
10.1.1.2
F0/0
R1
F0/1
C
10.3.3.3
S0/0/0
S0/0/1
R2
F0/0
S1
ACL 1 Pseudocode
If Source = 10.1.1.1 Permit
If Source = 10.1.1.x Deny
If Source = 10.x.x.x Permit

Figure 25-4 *Backdrop for Discussion of List Process with IP ACLs*

Consider the first-match ACL logic for a packet sent by host A to server S1. The source IP address will be 10.1.1.1, and it will be routed so that it enters R2's S0/0/1 interface, driving R2's ACL 1 logic. R2 compares this packet to the ACL, matching the first item in the list with a permit action. So this packet should be allowed through, as shown in Figure 25-5, on the left.

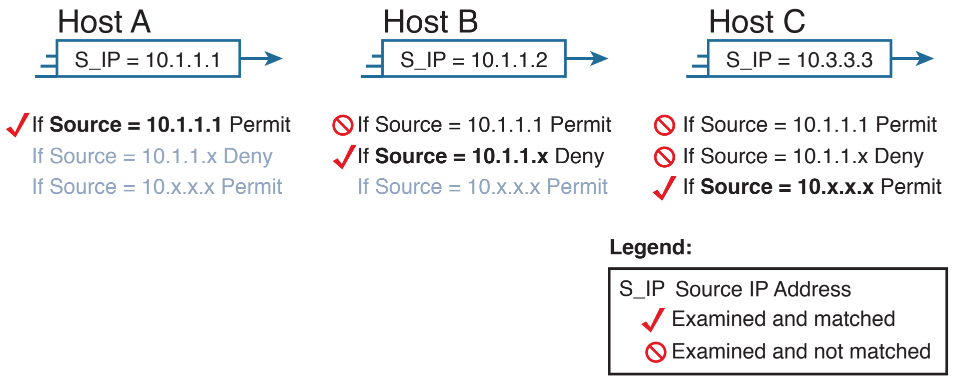

Host A
S_IP = 10.1.1.1
✓ If Source = 10.1.1.1 Permit
If Source = 10.1.1.x Deny
If Source = 10.x.x.x Permit
Host B
S_IP = 10.1.1.2
⃠ If Source = 10.1.1.1 Permit
✓ If Source = 10.1.1.x Deny
If Source = 10.x.x.x Permit
Host C
S_IP = 10.3.3.3
⃠ If Source = 10.1.1.1 Permit
⃠ If Source = 10.1.1.x Deny
✓ If Source = 10.x.x.x Permit
Legend:
S_IP Source IP Address
✓ Examined and matched
⃠ Examined and not matched

Figure 25-5 *ACL Items Compared for Packets from Hosts A, B, and C in Figure 25-4*

Next, consider a packet sent by host B, source IP address 10.1.1.2. When the packet enters R2's S0/0/1 interface, R2 compares the packet to ACL 1's first statement, and does not make a match (10.1.1.1 is not equal to 10.1.1.2). R2 then moves to the second statement, which requires some clarification. The ACL pseudocode, back in Figure 25-4, shows 10.1.1.x, which is meant to be shorthand that any value can exist in the last octet. Comparing only the first three octets, R2 decides that this latest packet does have a source IP address that begins with first three octets 10.1.1, so R2 considers that to be a match on the second statement. R2 takes the listed action (deny), discarding the packet. R2 also stops ACL processing on the packet, ignoring the third line in the ACL.

Finally, consider a packet sent by host C, again to server S1. The packet has source IP address 10.3.3.3, so when it enters R2's S0/0/1 interface, and drives ACL processing on R2, R2 looks at the first command in ACL 1. R2 does not match the first ACL command (10.1.1.1 in the command is not equal to the packet's 10.3.3.3). R2 looks at the second command, compares the first three octets (10.1.1) to the packet source IP address (10.3.3), still no match. R2 then looks at the third command. In this case, the wildcard means ignore the last three octets, and just compare the first octet (10), so the packet matches. R2 then takes the listed action (permit), allowing the packet to keep going.

This sequence of processing an ACL as a list happens for any type of IOS ACL: IP, other protocols, standard or extended, named or numbered.

Finally, if a packet does not match any of the items in the ACL, the packet is discarded. The reason is that every IP ACL has a *deny all* statement implied at the end of the ACL. It does not exist in the configuration, but if a router keeps searching the list, and no match is made by the end of the list, IOS considers the packet to have matched an entry that has a **deny** action.

Matching Logic and Command Syntax

Standard numbered IP ACLs use the following global command:

```
access-list {1-99 | 1300-1999} {permit | deny} matching-parameters
```

Each standard numbered ACL has one or more **access-list** commands with the same number, any number from the ranges shown in the preceding line of syntax. (One number is no better than the other.)

Besides the ACL number, each **access-list** command also lists the action (**permit** or **deny**), plus the matching logic. The rest of this section examines how to configure the matching parameters, which for standard ACLs, means that you can only match the source IP address, or portions of the source IP address using something called an ACL wildcard mask.

Matching the Exact IP Address

To match a specific source IP address, the entire IP address, all you have to do is type that IP address at the end of the command. For example, the previous example uses pseudocode for "permit if source = 10.1.1.1." The following command configures that logic with correct syntax using ACL number 1:

```
access-list 1 permit 10.1.1.1
```

Matching the exact full IP address is that simple.

In earlier IOS versions, the syntax included a **host** keyword. Instead of simply typing the full IP address, you first typed the **host** keyword, and then the IP address. Note that in later

IOS versions, if you use the **host** keyword, IOS accepts the command, but then removes the keyword.

```
access-list 1 permit host 10.1.1.1
```

Matching a Subset of the Address with Wildcards

Often, the business goals you want to implement with an ACL do not match a single particular IP address, but rather a range of IP addresses. Maybe you want to match all IP addresses in a subnet. Maybe you want to match all IP addresses in a range of subnets. Regardless, you want to check for more than one IP address in a range of addresses.

IOS allows standard ACLs to match a range of addresses using a tool called a *wildcard mask*. Note that this is not a subnet mask. The wildcard mask (which this book abbreviates as *WC mask*) gives the engineer a way to tell IOS to ignore parts of the address when making comparisons, essentially treating those parts as wildcards, as if they already matched.

You can think about WC masks in decimal and in binary, and both have their uses. To begin, think about WC masks in decimal, using these rules:

Key Topic

Decimal 0: The router must compare this octet as normal.

Decimal 255: The router ignores this octet, considering it to already match.

Keeping these two rules in mind, consider Figure 25-6, which demonstrates this logic using three different but popular WC masks: one that tells the router to ignore the last octet, one that tells the router to ignore the last two octets, and one that tells the router to ignore the last three octets.

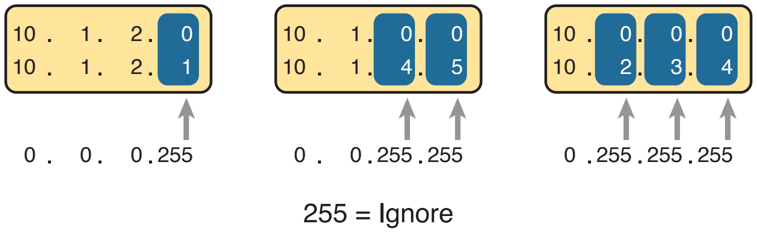


Figure 25-6 *Logic for WC Masks 0.0.0.255, 0.0.255.255, and 0.255.255.255*

All three examples in the boxes of Figure 25-6 show two numbers that are clearly different. The WC mask causes IOS to compare only some of the octets, while ignoring other octets. All three examples result in a match, because each wildcard mask tells IOS to ignore some octets. The example on the left shows WC mask 0.0.0.255, which tells the router to treat the last octet as a wildcard, essentially ignoring that octet for the comparison. Similarly, the middle example shows WC mask 0.0.255.255, which tells the router to ignore the two octets on the right. The rightmost case shows WC mask 0.255.255.255, telling the router to ignore the last three octets when comparing values.

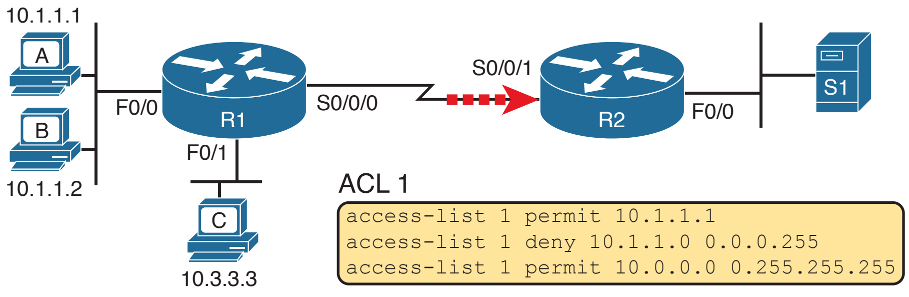
To see the WC mask in action, think back to the earlier example related to Figure 25-4 and Figure 25-5. The pseudocode ACL in those two figures used logic that can be created using a WC mask. As a reminder, the logic in the pseudocode ACL in those two figures included the following:

Line 1: Match and permit all packets with a source address of exactly 10.1.1.1.

Line 2: Match and deny all packets with source addresses with first three octets 10.1.1.

Line 3: Match and permit all addresses with first single octet 10.

Figure 25-7 shows the updated version of Figure 25-4, but with the completed, correct syntax, including the WC masks. In particular, note the use of WC mask 0.0.0.255 in the second command, telling R2 to ignore the last octet of the number 10.1.1.0, and the WC mask 0.255.255.255 in the third command, telling R2 to ignore the last three octets in the value 10.0.0.0.



```
access-list 1 permit 10.1.1.1
access-list 1 deny 10.1.1.0 0.0.0.255
access-list 1 permit 10.0.0.0 0.255.255.255
```

Figure 25-7 *Syntactically Correct ACL Replaces Pseudocode from Figure 25-4*

Finally, note that when using a WC mask, the **access-list** command's loosely defined *source* parameter should be a 0 in any octets where the WC mask is a 255. IOS will specify a source address to be 0 for the parts that will be ignored, even if nonzero values were configured.

Binary Wildcard Masks

Wildcard masks, as dotted-decimal number (DDN) values, actually represent a 32-bit binary number. As a 32-bit number, the WC mask actually directs the router's logic bit by bit. In short, a WC mask bit of 0 means the comparison should be done as normal, but a binary 1 means that the bit is a wildcard, and can be ignored when comparing the numbers.

Thankfully, for the purposes of CCENT and CCNA R&S study, and for most real-world applications, you can ignore the binary WC mask. Why? Well, we generally want to match a range of addresses that can be easily identified by a subnet number and mask, whether it be a real subnet, or a summary route that groups subnets together. (See DVD Appendix O, "Route Summarization," for more on summary routes.) If you can describe the range of addresses with a subnet number and mask, you can find the numbers to use in your ACL with some simple decimal math, as discussed next.

NOTE If you really want to know the binary mask logic, take the two DDN numbers the ACL will compare (one from the **access-list** command, and the other from the packet header) and convert both to binary. Then, also convert the WC mask to binary. Compare the first two binary numbers bit by bit, but also ignore any bits for which the WC mask happens to list a binary 1, because that tells you to ignore the bit. If all the bits you checked are equal, it's a match!

Finding the Right Wildcard Mask to Match a Subnet

In many cases, an ACL needs to match all hosts in a particular subnet. To match a subnet with an ACL, you can use the following shortcut:

Key Topic

- Use the subnet number as the source value in the **access-list** command.
- Use a wildcard mask found by subtracting the subnet mask from 255.255.255.255.

For example, for subnet 172.16.8.0 255.255.252.0, use the subnet number (172.16.8.0) as the address parameter, and then do the following math to find the wildcard mask:

$$\begin{array}{r} 255.255.255.255 \\ -255.255.252.0 \\ \hline 0.\quad 0.\quad 3.255 \end{array}$$

Continuing this example, a completed command for this same subnet would be as follows:

```
access-list 1 permit 172.16.8.0 0.0.3.255
```

The upcoming section, "Practice Applying Standard IP ACLs," gives you a chance to practice matching subnets when configuring ACLs.

Matching Any/All Addresses

In some cases, you will want one ACL command to match any and all packets that reach that point in the ACL. First, you have to know the (simple) way to match all packets using the **any** keyword. More importantly, you need to think about when to match any and all packets.

First, to match any and all packets with an ACL command, just use the **any** keyword for the address. For example, to permit all packets:

```
access-list 1 permit any
```

So, when and where should you use such a command? Remember, all Cisco IP ACLs end with an implicit deny any concept at the end of each ACL. That is, if a router compares a packet to the ACL, and the packet matches none of the configured statements, the router discards the packet. Want to override that default behavior? Configure a **permit any** at the end of the ACL.

You might also want to explicitly configure a command to deny all traffic (for example, **access-list 1 deny any**) at the end of an ACL. Why, when the same logic already sits at the end of the ACL anyway? Well, the ACL **show** commands list counters for the number of packets matched by each command in the ACL, but there is no counter for that implicit deny any concept at the end of the ACL. So, if you want to see counters for how many packets are matched by the deny any logic at the end of the ACL, configure an explicit **deny any**.

Implementing Standard IP ACLs

This chapter has already introduced all the configuration steps in bits and pieces. This section summarizes those pieces as a configuration process. The process also refers to the **access-list** command, whose generic syntax is repeated here for reference:

```
access-list access-list-number {deny | permit} source [source-wildcard]
```

Key Topic

Step 1. Plan the location (router and interface) and direction (in or out) on that interface:

A. Standard ACLs should be placed near to the destination of the packets so that they do not unintentionally discard packets that should not be discarded.

B. Because standard ACLs can only match a packet's source IP address, identify the source IP addresses of packets as they go in the direction that the ACL is examining.

Step 2. Configure one or more **access-list** global configuration commands to create the ACL, keeping the following in mind:

A. The list is searched sequentially, using first-match logic.

B. The default action, if a packet does not match any of the **access-list** commands, is to **deny** (discard) the packet.

Step 3. Enable the ACL on the chosen router interface, in the correct direction, using the **ip access-group** *number* {**in** | **out**} interface subcommand.

The rest of this section shows a couple of examples.

Standard Numbered ACL Example 1

The first example shows the configuration for the same requirements demonstrated with Figure 25-4 and Figure 25-5. Restated, the requirements for this ACL are as follows:

1. Enable the ACL inbound on R2's S0/0/1 interface.
2. Permit packets coming from host A.
3. Deny packets coming from other hosts in host A's subnet.
4. Permit packets coming from any other address in Class A network 10.0.0.0.
5. The original example made no comment about what to do by default, so simply deny all other traffic.

Example 25-1 shows a completed correct configuration, starting with the configuration process, followed by output from the **show running-config** command.

Example 25-1 *Standard Numbered ACL Example 1 Configuration*

```
R2# configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
R2(config)# access-list 1 permit 10.1.1.1
R2(config)# access-list 1 deny 10.1.1.0 0.0.0.255
R2(config)# access-list 1 permit 10.0.0.0 0.255.255.255
R2(config)# interface S0/0/1
R2(config-if)# ip access-group 1 in
R2(config-if)# ^Z
R2# show running-config
! Lines omitted for brevity

access-list 1 permit 10.1.1.1
access-list 1 deny 10.1.1.0 0.0.0.255
access-list 1 permit 10.0.0.0 0.255.255.255
```

First, pay close attention to the configuration process at the top of the example. Note that the **access-list** command does not change the command prompt from the global configuration mode prompt, because the **access-list** command is a global configuration command. Then, compare that to the output of the **show running-config** command: the details are identical compared to the commands that were added in configuration mode. Finally, make sure to note the **ip access-group 1 in** command, under R2's S0/0/1 interface, which enables the ACL logic (both location and direction).

Example 25-2 lists some output from Router R2 that shows information about this ACL. The **show ip access-lists** command lists details about IPv4 ACLs only, while the **show access-lists** command lists details about IPv4 ACLs plus any other types of ACLs that are currently configured, for example, IPv6 ACLs.

Example 25-2 *ACL* **show** *Commands on R2*

```
R2# show ip access-lists
Standard IP access list 1
    10 permit 10.1.1.1 (107 matches)
    20 deny   10.1.1.0, wildcard bits 0.0.0.255 (4 matches)
    30 permit 10.0.0.0, wildcard bits 0.255.255.255 (10 matches)
R2# show access-lists
Standard IP access list 1
    10 permit 10.1.1.1 (107 matches)
    20 deny   10.1.1.0, wildcard bits 0.0.0.255 (4 matches)
    30 permit 10.0.0.0, wildcard bits 0.255.255.255 (10 matches)
R2# show ip interface s0/0/1
Serial0/0/1 is up, line protocol is up
  Internet address is 10.1.2.2/24
  Broadcast address is 255.255.255.255
  Address determined by setup command
  MTU is 1500 bytes
  Helper address is not set
  Directed broadcast forwarding is disabled
  Multicast reserved groups joined: 224.0.0.9
  Outgoing access list is not set
  Inbound  access list is 1
! Lines omitted for brevity
```

The output of these commands shows two items of note. The first line of output in this case notes the type (standard), and the number. If more than one ACL existed, you would see multiple stanzas of output, one per ACL, each with a heading line like this one. Next, these commands list packet counts for the number of packets that the router has matched with each command. For example, 107 packets so far have matched the first line in the ACL.

Finally, the end of the example lists the **show ip interface** command output. This command lists, among many other items, the number or name of any IP ACL enabled on the interface per the **ip access-group** interface subcommand.

Standard Numbered ACL Example 2

For the second example, use Figure 25-8, and imagine your boss gave you some requirements hurriedly in the hall. At first, he tells you he wants to filter packets going from the servers on the right toward the clients on the left. Then, he says he wants you to allow access for hosts A, B, and other hosts in their same subnet to server S1, but deny access to that server to the hosts in host C's subnet. Then, he tells you that, additionally, hosts in host A's subnet should be denied access to server S2, but hosts in host C's subnet should be allowed access to server S2. All by filtering packets going right to left only, and then he tells you: put the ACL inbound on R2's F0/0 interface.

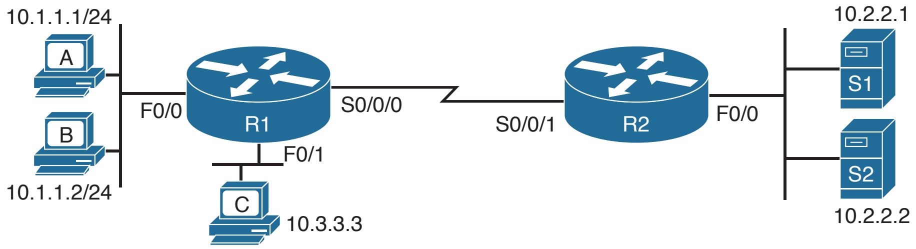


Figure 25-8 *Standard Numbered ACL Example 2*

If you cull through all the boss's comments, the requirements might reduce to the following:

1. Enable the ACL inbound on R2's F0/0 interface.
2. Permit packets from server S1 going to hosts in A's subnet.
3. Deny packets from server S1 going to hosts in C's subnet.
4. Permit packets from server S2 going to hosts in C's subnet.
5. Deny packets from server S2 going to hosts in A's subnet.
6. (There was no comment about what to do by default; use the implied **deny all** default.)

As it turns out, you cannot do everything your boss asked with a standard ACL. For example, consider the obvious command for requirement number 2: **access-list 2 permit 10.2.2.1**. That permits all traffic whose source IP is 10.2.2.1 (server S1). The very next requirement asks you to filter (deny) packets sourced from that same IP address! Even if you added another command that checked for source IP address 10.2.2.1, the router would never get to it, because routers use first-match logic when searching the ACL. You cannot check both the destination and source IP address, because standard ACLs cannot check the destination IP address.

To solve this problem, you should get a new boss! No, seriously, you have to rethink the problem and change the rules. In real life, you would probably use an extended ACL instead, which lets you check both the source and destination IP address.

For the sake of practicing another standard ACL, imagine your boss lets you change the requirements. First, you will use two outbound ACLs, both on Router R1. Each ACL will permit traffic from a single server to be forwarded onto that connected LAN, with the following modified requirements:

1. Using an outbound ACL on R1's F0/0 interface, permit packets from server S1, and deny all other packets.
2. Using an outbound ACL on R1's F0/1 interface, permit packets from server S2, and deny all other packets.

Example 25-3 shows the configuration that completes these requirements.

Example 25-3 *Alternative Configuration in Router R1*

```
access-list 2 remark This ACL permits server S1 traffic to host A's subnet
access-list 2 permit 10.2.2.1
!
access-list 3 remark This ACL permits server S2 traffic to host C's subnet
access-list 3 permit 10.2.2.2
!
interface F0/0
 ip access-group 2 out
!
interface F0/1
 ip access-group 3 out
```

As highlighted in the example, the solution with ACL number 2 permits all traffic from server S1, with that logic enabled for packets exiting R1's F0/0 interface. All other traffic will be discarded because of the implied deny all at the end of the ACL. In addition, ACL 3 permits traffic from server S2, which is then permitted to exit R1's F0/1 interface. Also, note that the solution shows the use of the **access-list remark** parameter, which allows you to leave text documentation that stays with the ACL.

NOTE When routers apply an ACL to filter packets in the outbound direction, as shown in Example 25-3, the router checks packets that it routes against the ACL. However, a router does not filter packets that the router itself creates with an outbound ACL. Examples of those packets include routing protocol messages, and packets sent by the **ping** and **traceroute** commands on that router.

Troubleshooting and Verification Tips

Troubleshooting IPv4 ACLs requires some attention to detail. In particular, you have to be ready to look at the address and wildcard mask and confidently predict the addresses matched by those two combined parameters. The upcoming practice problems a little later in this chapter can help prepare you for that part of the work. But a few other tips can help you verify and troubleshoot ACL problems on the exams as well.

First, you can tell if the router is matching packets or not with a couple of tools. Example 25-2 already showed that IOS keeps statistics about the packets matched by each line of an ACL. In addition, if you add the **log** keyword to the end of an **access-list** command, IOS then issues log messages with occasional statistics about matches of that particular line of the ACL. Both the statistics and the log messages can be helpful in deciding which line in the ACL is being matched by a packet.

For example, Example 25-4 shows an updated version of ACL 2 from Example 25-3, this time with the **log** keyword added. The bottom of the example then shows a typical log message, this one showing the resulting match based on a packet with source IP address 10.2.2.1 (as matched with the ACL), to destination address 10.1.1.1.

Example 25-4 *Creating Log Messages for ACL Statistics*

```
R1# show running-config
! lines removed for brevity
access-list 2 remark This ACL permits server S1 traffic to host A's subnet
access-list 2 permit 10.2.2.1 log
!
interface F0/0
 ip access-group 2 out


R1#
Feb  4 18:30:24.082: %SEC-6-IPACCESSLOGNP: list 2 permitted 0 10.2.2.1 -> 10.1.1.1, 1
  packet
```

Anytime you troubleshoot an ACL for the first time, before getting into the details of the matching logic, take the time to think about both the interface on which the ACL is enabled, and the direction of packet flow. Sometimes, the matching logic is perfect—but the ACL has been enabled on the wrong interface, or for the wrong direction, to match the packets as configured for the ACL.

For example, Figure 25-9 repeats the same ACL shown earlier in Figure 25-7. The first line of that ACL matches the specific host address 10.1.1.1. If that ACL exists on Router R2, placing that ACL as an inbound ACL on R2's S0/0/1 interface can work, because packets sent by host 10.1.1.1—on the left side of the figure—can enter R2's S0/0/1 interface. However, if R2 enables ACL 1 on its F0/0 interface, for inbound packets, the ACL will never match a packet with source IP address 10.1.1.1, because packets sent by host 10.1.1.1 will never enter that interface. Packets sent by 10.1.1.1 will exit R2's F0/0 interface, but never enter it, just because of the network topology.

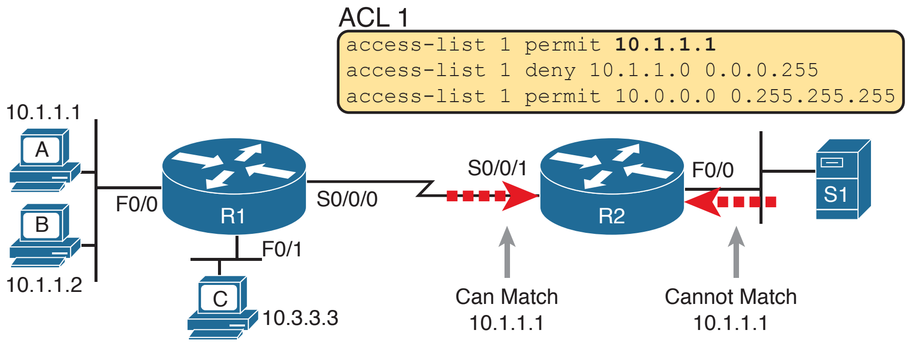

ACL 1
access-list 1 permit 10.1.1.1
access-list 1 deny 10.1.1.0 0.0.0.255
access-list 1 permit 10.0.0.0 0.255.255.255
10.1.1.1
A
B
10.1.1.2
F0/0
R1
S0/0/0
F0/1
C
10.3.3.3
S0/0/1
R2
F0/0
S1
Can Match
10.1.1.1
Cannot Match
10.1.1.1

Figure 25-9 *Example of Checking the Interface and Direction for an ACL*

Practice Applying Standard IP ACLs

Some CCENT and CCNA R&S topics, like ACLs, simply require more drills and practice than others. ACLs require you to think of parameters to match ranges of numbers, and that of course requires some use of math, and some use of processes.

This section provides some practice problems and tips, from two perspectives. First, this section asks you to build one-line standard ACLs to match some packets. Second, this section asks you to interpret existing ACL commands to describe what packets the ACL will match. Both skills are useful for the exams.

Practice Building access-list Commands

In this section, practice getting comfortable with the syntax of the **access-list** command, particularly with choosing the correct matching logic. These skills will be helpful when reading about extended and named ACLs in the next chapter.

First, the following list summarizes some important tips to consider when choosing matching parameters to any **access-list** command:

Key Topic

- To match a specific address, just list the address.
- To match any and all addresses, use the **any** keyword.
- To match based only on the first one, two, or three octets of an address, use the 0.255.255.255, 0.0.255.255, and 0.0.0.255 WC masks, respectively. Also, make the source (address) parameter have 0s in the wildcard octets (those octets with 255 in the wildcard mask).
- To match a subnet, use the subnet ID as the source, and find the WC mask by subtracting the DDN subnet mask from 255.255.255.255.

Table 25-2 lists the criteria for several practice problems. Your job: Create a one-line standard ACL that matches the packets. The answers are listed in the section, "Answers to Earlier Practice Problems," later in this chapter.

Table 25-2 Building One-Line Standard ACLs: Practice

Problem	Criteria
1	Packets from 172.16.5.4
2	Packets from hosts with 192.168.6 as the first three octets
3	Packets from hosts with 192.168 as the first two octets
4	Packets from any host
5	Packets from subnet 10.1.200.0/21
6	Packets from subnet 10.1.200.0/27
7	Packets from subnet 172.20.112.0/23
8	Packets from subnet 172.20.112.0/26
9	Packets from subnet 192.168.9.64/28
10	Packets from subnet 192.168.9.64/30

Reverse Engineering from ACL to Address Range

In some cases, you may not be creating your own ACL. Instead, you may need to interpret some existing **access-list** commands. To answer these types of questions on the exams, you need to determine the range of IP addresses matched by a particular address/wildcard mask combination in each ACL statement.

Under certain assumptions that are reasonable for CCENT and CCNA R&S certifications, calculating the range of addresses matched by an ACL can be relatively simple. Basically, the range of addresses begins with the address configured in the ACL command. The range of addresses ends with the sum of the address field and the wildcard mask. That's it.

For example, with the command **access-list 1 permit 172.16.200.0 0.0.7.255**, the low end of the range is simply 172.16.200.0, taken directly from the command itself. Then, to find the high end of the range, just add this number to the WC mask, as follows:

```
  172.16.200.0
+ 0.  0.   7.255
----------------
  172.16.207.255
```

For this last bit of practice, look at the existing **access-list** commands in Table 25-3. In each case, make a notation about the exact IP address, or range of IP addresses, matched by the command.

Table 25-3 Finding IP Addresses/Ranges Matching by Existing ACLs

Problem	Commands for Which to Predict the Source Address Range
1	**access-list 1 permit 10.7.6.5**
2	**access-list 2 permit 192.168.4.0 0.0.0.127**
3	**access-list 3 permit 192.168.6.0 0.0.0.31**
4	**access-list 4 permit 172.30.96.0 0.0.3.255**
5	**access-list 5 permit 172.30.96.0 0.0.0.63**
6	**access-list 6 permit 10.1.192.0 0.0.0.31**
7	**access-list 7 permit 10.1.192.0 0.0.1.255**
8	**access-list 8 permit 10.1.192.0 0.0.63.255**

Interestingly, IOS lets the CLI user type an **access-list** command in configuration mode, and IOS will potentially change the address parameter before placing the command into the running-config file. This process of just finding the range of addresses matched by the **access-list** command expects that the **access-list** command came from the router, so that any such changes were complete.

The change IOS can make with an **access-list** command is to convert to 0 any octet of an address for which the wildcard mask's octet is 255. For example, with a wildcard mask of 0.0.255.255, IOS ignores the last two octets. IOS expects the address field to end with two 0s. If not, IOS still accepts the **access-list** command, but IOS changes the last two octets of the address to 0s. Example 25-5 shows an example, where the configuration shows address 10.1.1.1, but wildcard mask 0.0.255.255.

Example 25-5 *IOS Changing the Address Field in an* **access-list** *Command*

```
R2# configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
R2(config)# access-list 21 permit 10.1.1.1 0.0.255.255
R2(config)# ^Z
R2#
R2# show ip access-lists
Standard IP access list 21
    10 permit 10.1.0.0, wildcard bits 0.0.255.255
```

The math to find the range of addresses relies on the fact that either the command is fully correct, or that IOS has already set these address octets to 0, as shown in the example.

NOTE The most useful WC masks, in binary, do not interleave 0s and 1s. This book assumes the use of only these types of WC masks. However, Cisco IOS allows WC masks that interleave 0s and 1s, but using these WC masks break the simple method of calculating the range of addresses. As you progress through to CCIE studies, be ready to dig deeper to learn how to determine what an ACL matches.

Chapter Review

One key to doing well on the exams is to perform repetitive spaced review sessions. Review this chapter's material using either the tools in the book, DVD, or interactive tools for the same material found on the book's companion website. Refer to the "Your Study Plan" element for more details. Table 25-4 outlines the key review elements and where you can find them. To better track your study progress, record when you completed these activities in the second column.

Table 25-4 Chapter Review Tracking

Review Element	Review Date(s)	Resource Used
Review key topics		Book, DVD/website
Review key terms		Book, DVD/website
Repeat DIKTA questions		Book, PCPT
Review command tables		Book

Review All the Key Topics

Key Topic

Table 25-5 Key Topics for Chapter 25

Key Topic Element	Description	Page Number
Paragraph	Summary of the general rule of the location and direction for an ACL	595
Figure 25-3	Summary of four main categories of IPv4 ACLs in Cisco IOS	597
Paragraph	Summary of first-match logic used by all ACLs	598
List	Wildcard mask logic for decimal 0 and 255	600
List	Wildcard mask logic to match a subnet	602
List	Steps to plan and implement a standard IP ACL	603
List	Tips for creating matching logic for the source address field in the **access-list** command	608

Key Terms You Should Know

standard access list, wildcard mask

Additional Practice for This Chapter's Processes

For additional practice with analyzing subnets, you may do the same set of practice problems using your choice of tools:

Application: Use the Basic IPv4 Access Control Lists application on the DVD or companion website.

PDF: Alternatively, practice the same problems found in these apps using DVD Appendix I, "Practice for Chapter 25: Basic IPv4 Access Control Lists."

Command References

Tables 25-6 and 25-7 list configuration and verification commands used in this chapter. As an easy review exercise, cover the left column in a table, read the right column, and try to recall the command without looking. Then repeat the exercise, covering the right column, and try to recall what the command does.

Table 25-6 Chapter 25 Configuration Command Reference

Command	Description
access-list *access-list-number* {**deny** | **permit**} *source* [*source-wildcard*] [**log**]	Global command for standard numbered access lists. Use a number between 1 and 99 or 1300 and 1999, inclusive
access-list *access-list-number* **remark** *text*	Defines a remark that helps you remember what the ACL is supposed to do
ip access-group *number* {**in** | **out**}	Interface subcommand to enable access lists

Table 25-7 Chapter 25 EXEC Command Reference

Command	Description
show ip interface [*type number*]	Includes a reference to the access lists enabled on the interface
show access-lists [*access-list-number* | *access-list-name*]	Shows details of configured access lists for all protocols
show ip access-lists [*access-list-number* | *access-list-name*]	Shows IP access lists

Answers to Earlier Practice Problems

Table 25-8 lists the answers to the problems listed earlier in Table 25-2.

Table 25-8 Building One-Line Standard ACLs: Answers

Problem	Answers
1	access-list 1 permit 172.16.5.4
2	access-list 2 permit 192.168.6.0 0.0.0.255
3	access-list 3 permit 192.168.0.0 0.0.255.255
4	access-list 4 permit any
5	access-list 5 permit 10.1.200.0 0.0.7.255
6	access-list 6 permit 10.1.200.0 0.0.0.31
7	access-list 7 permit 172.20.112.0 0.0.1.255
8	access-list 8 permit 172.20.112.0 0.0.0.63
9	access-list 9 permit 192.168.9.64 0.0.0.15
10	access-list 10 permit 192.168.9.64 0.0.0.3

Table 25-9 lists the answers to the problems listed earlier in Table 25-3.

Table 25-9 Address Ranges for Problems in Table 25-3: Answers

Problem	Address Range
1	One address: 10.7.6.5
2	192.168.4.0 – 192.168.4.127
3	192.168.6.0 – 192.168.6.31
4	172.30.96.0 – 172.30.99.255
5	172.30.96.0 – 172.30.96.63
6	10.1.192.0 – 10.1.192.31
7	10.1.192.0 – 10.1.193.255
8	10.1.192.0 – 10.1.255.255

This page intentionally left blank

CHAPTER 26

Advanced IPv4 Access Control Lists

This chapter covers the following exam topics:

4.0 Infrastructure Services

4.6 Configure, verify, and troubleshoot IPv4 standard numbered and named access list for routed interfaces

Cisco routers use IPv4 access control lists (ACL) for many different applications: to match packets to make filtering decisions, to match packets for Network Address Translation (NAT), to match packets to make quality of service (QoS) decisions, and for several other reasons.

IPv4 ACLs are either standard or extended ACLs, with standard ACLs matching only the source IP address, and extended matching a variety of packet header fields. At the same time, IP ACLs are either numbered or named. Figure 26-1 shows the categories, and the main features of each, as introduced in the previous chapter.

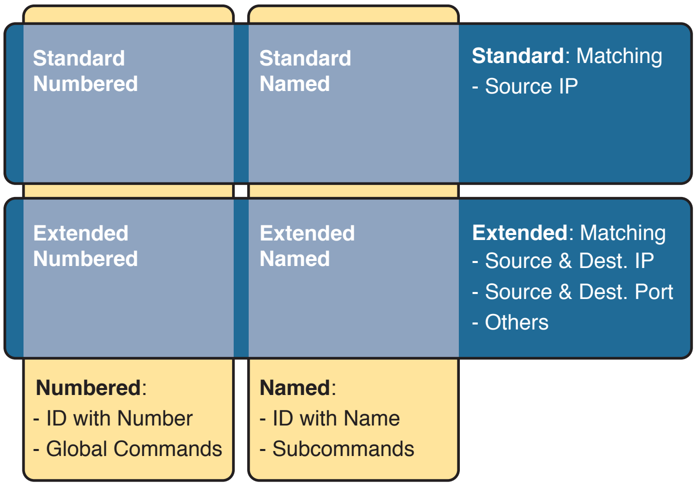


Figure 26-1 *Comparisons of IP ACL Types*

This chapter discusses the other three categories of ACLs beyond standard numbered IP ACLs, and ends with a few miscellaneous features to secure Cisco routers and switches.

"Do I Know This Already?" Quiz

Take the quiz (either here, or use the PCPT software) if you want to use the score to help you decide how much time to spend on this chapter. The answers are at the bottom of the page following the quiz, and the explanations are in DVD Appendix C and in the PCPT software.

Table 26-1 "Do I Know This Already?" Foundation Topics Section-to-Question Mapping

Foundation Topics Section	Questions
Extended IP Access Control Lists	1–3
Named ACLs and ACL Editing	4
Troubleshooting with IPv4 ACLs	5–6

1. Which of the following fields cannot be compared based on an extended IP ACL? (Choose two answers.)

a. Protocol

b. Source IP address

c. Destination IP address

d. TOS byte

e. URL

f. Filename for FTP transfers

2. Which of the following **access-list** commands permit packets going from host 10.1.1.1 to all web servers whose IP addresses begin with 172.16.5? (Choose two answers.)

a. **access-list 101 permit tcp host 10.1.1.1 172.16.5.0 0.0.0.255 eq www**

b. **access-list 1951 permit ip host 10.1.1.1 172.16.5.0 0.0.0.255 eq www**

c. **access-list 2523 permit ip host 10.1.1.1 eq www 172.16.5.0 0.0.0.255**

d. **access-list 2523 permit tcp host 10.1.1.1 eq www 172.16.5.0 0.0.0.255**

e. **access-list 2523 permit tcp host 10.1.1.1 172.16.5.0 0.0.0.255 eq www**

3. Which of the following **access-list** commands permits packets going to any web client from all web servers whose IP addresses begin with 172.16.5?

a. **access-list 101 permit tcp host 10.1.1.1 172.16.5.0 0.0.0.255 eq www**

b. **access-list 1951 permit ip host 10.1.1.1 172.16.5.0 0.0.0.255 eq www**

c. **access-list 2523 permit tcp any eq www 172.16.5.0 0.0.0.255**

d. **access-list 2523 permit tcp 172.16.5.0 0.0.0.255 eq www 172.16.5.0 0.0.0.255**

e. **access-list 2523 permit tcp 172.16.5.0 0.0.0.255 eq www any**

4. In a router running a recent IOS version (at least version 15.0), an engineer needs to delete the second line in ACL 101, which currently has four commands configured. Which of the following options could be used? (Choose two answers.)
   a. Delete the entire ACL and reconfigure the three ACL statements that should remain in the ACL.
   b. Delete one line from the ACL using the **no access-list...** global command.
   c. Delete one line from the ACL by entering ACL configuration mode for the ACL and then deleting only the second line based on its sequence number.
   d. Delete the last three lines from the ACL from global configuration mode, and then add the last two statements back into the ACL.
5. An engineer is considering configuring an ACL on Router R1. The engineer could use ACL A which would be enabled with the **ip access-group A out** command on interface G0/1, or ACL B, which would be enabled with the **ip access-group B in** command on that same interface. R1's G0/1 interface uses IPv4 address 1.1.1.1. Which of the answers is true when comparing these options? (Choose two answers.)
   a. ACL A creates more risk of filtering important overhead traffic than ACL B.
   b. ACL B creates more risk of filtering important overhead traffic than ACL A.
   c. A **ping 1.1.1.1** command on R1 would bypass ACL A even if enabled.
   d. A **ping 1.1.1.1** command on R1 would bypass ACL B even if enabled.
6. An engineer configures an ACL but forgets to save the configuration. At that point, which of the following commands displays the configuration of an IPv4 ACL, including line numbers? (Choose two answers.)
   a. **show running-config**
   b. **show startup-config**
   c. **show ip access-lists**
   d. **show access-lists**

Foundation Topics

Extended Numbered IP Access Control Lists

Extended IP access lists have many similarities compared to the standard numbered IP ACLs discussed in the previous chapter. Just like standard IP ACLs, you enable extended access lists on interfaces for packets either entering or exiting the interface. IOS searches the list sequentially. Extended ACLs also use first-match logic, because the router stops the search through the list as soon as the first statement is matched, taking the action defined in the first-matched statement. All these features are also true of standard numbered access lists (and named ACLs).

Extended ACLs differ from standard ACLs mostly because of the larger variety of packet header fields that can be used to match a packet. One extended ACL statement can examine multiple parts of the packet headers, requiring that all the parameters be matched correctly to match that one ACL statement. That powerful matching logic makes extended access lists both more useful and more complex than standard IP ACLs.

Matching the Protocol, Source IP, and Destination IP

Like standard numbered IP ACLs, extended numbered IP ACLs also use the **access-list** global command. The syntax is identical, at least up through the **permit** or **deny** keyword. At that point, the command lists matching parameters, and those differ, of course. In particular, the extended ACL **access-list** command requires three matching parameters: the IP protocol type, the source IP address, and the destination IP address.

The IP header's Protocol field identifies the header that follows the IP header. Figure 26-2 shows the location of the IP Protocol field, the concept of it pointing to the type of header that follows, along with some details of the IP header for reference.

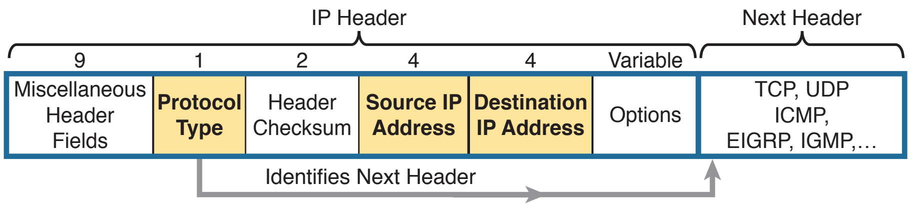


Figure 26-2 *IP Header, with Focus on Required Fields in Extended IP ACLs*

IOS requires that you configure parameters for the three highlighted parts of Figure 26-2. For the protocol type, you simply use a keyword, such as **tcp**, **udp**, or **icmp**, matching IP packets that happen to have a TCP, UDP, or ICMP header, respectively, following the IP header. Or you can use the keyword **ip**, which means "all IPv4 packets." You also must configure some values for the source and destination IP address fields that follow; these fields use the same syntax and options for matching the IP addresses as discussed in Chapter 25, "Basic IPv4 Access Control Lists." Figure 26-3 shows the syntax.

Key Topic

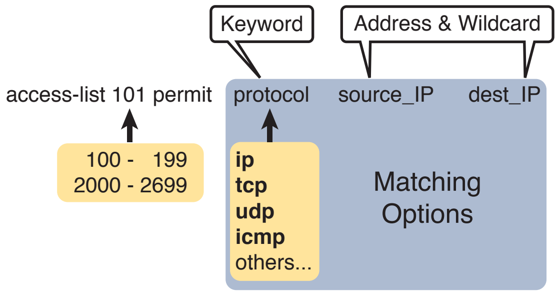


Figure 26-3 *Extended ACL Syntax, with Required Fields*

NOTE When matching IP addresses in the source and destination fields, there is one difference with standard ACLs: When matching a specific IP address, the extended ACL requires the use of the **host** keyword. You cannot simply list the IP address alone.

Answers to the "Do I Know This Already?" quiz:

1 E, F **2** A, E **3** E **4** A, C **5** B, C **6** C, D

Table 26-2 lists several sample **access-list** commands that use only the required matching parameters. Feel free to cover the right side and use the table for an exercise, or just review the explanations to get an idea for the logic in some sample commands.

Table 26-2 Extended **access-list** Commands and Logic Explanations

access-list Statement	What It Matches
access-list 101 deny tcp any any	Any IP packet that has a TCP header
access-list 101 deny udp any any	Any IP packet that has a UDP header
access-list 101 deny icmp any any	Any IP packet that has an ICMP header
access-list 101 deny ip host 1.1.1.1 host 2.2.2.2	All IP packets from host 1.1.1.1 going to host 2.2.2.2, regardless of the header after the IP header
access-list 101 deny udp 1.1.1.0 0.0.0.255 any	All IP packets that have a UDP header following the IP header, from subnet 1.1.1.0/24, and going to any destination

The last entry in Table 26-2 helps make an important point about how IOS processes extended ACLs:

Key Topic

In an extended ACL **access-list** command, all the matching parameters must match the packet for the packet to match the command.

For example, in that last example from Table 26-2, the command checks for UDP, a source IP address from subnet 1.1.1.0/24, and any destination IP address. If a packet with source IP address 1.1.1.1 were examined, it would match the source IP address check, but if it had a TCP header instead of UDP, it would not match this **access-list** command. All parameters must match.

Matching TCP and UDP Port Numbers

Extended ACLs can also examine parts of the TCP and UDP headers, particularly the source and destination port number fields. The port numbers identify the application that sends or receives the data.

The most useful ports to check are the well-known ports used by servers. For example, web servers use well-known port 80 by default. Figure 26-4 shows the location of the port numbers in the TCP header, following the IP header.

Key Topic

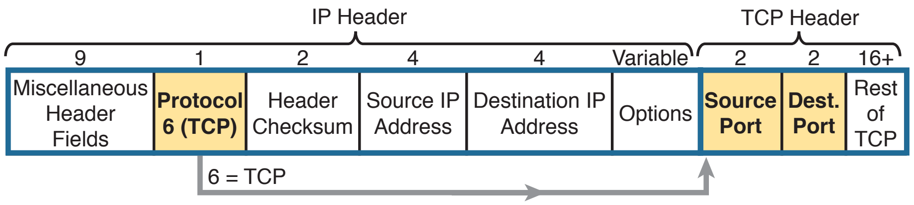


Figure 26-4 *IP Header, Followed by a TCP Header and Port Number Fields*

When an extended ACL command includes either the **tcp** or **udp** keyword, that command can optionally reference the source and/or destination port. To make these comparisons, the syntax uses keywords for equal, not equal, less than, greater than, and for a range of port numbers. In addition, the command can use either the literal decimal port numbers, or more convenient keywords for some well-known application ports. Figure 26-5 shows the positions of the source and destination port fields in the **access-list** command and these port number keywords.

Key Topic

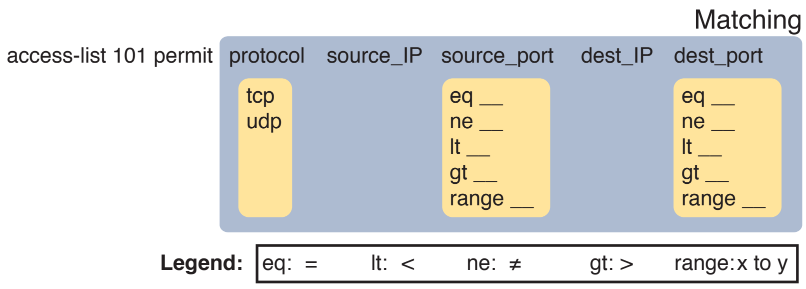


Figure 26-5 *Extended ACL Syntax with TCP and UDP Port Numbers Enabled*

For example, consider the simple network shown in Figure 26-6. The FTP server sits on the right, with the client on the left. The figure shows the syntax of an ACL that matches the following:

- Packets that include a TCP header
- Packets sent from the client subnet
- Packets sent to the server subnet
- Packets with TCP destination port 21 (FTP server control port)

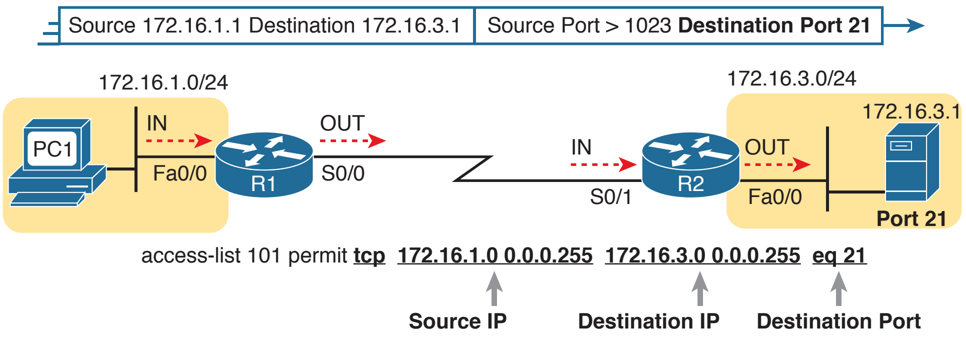

Source 172.16.1.1 Destination 172.16.3.1
Source Port > 1023 Destination Port 21
172.16.1.0/24
172.16.3.0/24
PC1
IN
Fa0/0
R1
OUT
S0/0
IN
S0/1
R2
OUT
Fa0/0
172.16.3.1
Port 21
access-list 101 permit tcp 172.16.1.0 0.0.0.255 172.16.3.0 0.0.0.255 eq 21
Source IP
Destination IP
Destination Port

Figure 26-6 *Filtering Packets Based on Destination Port*

To fully appreciate the matching of the destination port with the **eq 21** parameters, consider packets moving from left to right, from PC1 to the server. Assuming the server uses well-known port 21 (FTP control port), the packet's TCP header has a destination port value of 21. The ACL syntax includes the **eq 21** parameters after the destination IP address. The position after the destination address parameters is important: That position identifies the fact that the **eq 21** parameters should be compared to the packet's destination port. As a result, the ACL statement shown in Figure 26-6 would match this packet, and the destination port of 21, if used in any of the four locations implied by the four dashed arrowed lines in the figure.

Conversely, Figure 26-7 shows the reverse flow, with a packet sent by the server back toward PC1. In this case, the packet's TCP header has a source port of 21, so the ACL must check the source port value of 21, and the ACL must be located on different interfaces. In this case, the **eq 21** parameters follow the source address field, but come before the destination address field.

Key Topic

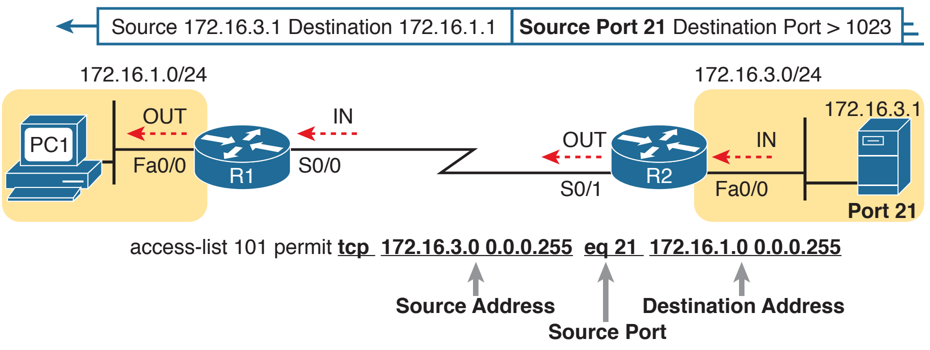

Source 172.16.3.1 Destination 172.16.1.1
Source Port 21 Destination Port > 1023
172.16.1.0/24
172.16.3.0/24
PC1
OUT
Fa0/0
R1
IN
S0/0
OUT
S0/1
R2
IN
Fa0/0
172.16.3.1
Port 21
access-list 101 permit tcp 172.16.3.0 0.0.0.255 eq 21 172.16.1.0 0.0.0.255
Source Address
Source Port
Destination Address

Figure 26-7 *Filtering Packets Based on Source Port*

When examining ACLs that match port numbers, first consider the location and direction in which the ACL will be applied. That direction determines whether the packet is being sent to the server, or from the server. At that point, you can decide whether you need to check the source or destination port in the packet. For reference, Table 26-3 lists many of the popular port numbers and their transport layer protocols and applications. Note that the syntax of the **access-list** commands accepts both the port numbers and a shorthand version of the application name.

Table 26-3 Popular Applications and Their Well-Known Port Numbers

Port Number(s)	Protocol	Application	access-list Command Keyword
20	TCP	FTP data	**ftp-data**
21	TCP	FTP control	**ftp**
22	TCP	SSH	—
23	TCP	Telnet	**telnet**
25	TCP	SMTP	**smtp**
53	UDP, TCP	DNS	**domain**
67	UDP	DHCP Server	—
68	UDP	DHCP Client	—
69	UDP	TFTP	**tftp**
80	TCP	HTTP (WWW)	**www**
110	TCP	POP3	**pop3**
161	UDP	SNMP	**snmp**
443	TCP	SSL	—
514	UDP	Syslog	—
16,384 – 32,767	UDP	RTP (voice, video)	—

Table 26-4 lists several example **access-list** commands that match based on port numbers. Cover the right side of the table, and try to characterize the packets matched by each command. Then, check the right side of the table to see if you agree with the assessment.

Table 26-4 Extended **access-list** Command Examples and Logic Explanations

access-list Statement	What It Matches
access-list 101 deny tcp any gt 1023 host 10.1.1.1 eq 23	Packets with a TCP header, any source IP address, with a source port greater than (gt) 1023, a destination IP address of exactly 10.1.1.1, and a destination port equal to (eq) 23.
access-list 101 deny tcp any host 10.1.1.1 eq 23	The same as the preceding example, but any source port matches, because that parameter is omitted in this case.
access-list 101 deny tcp any host 10.1.1.1 eq telnet	The same as the preceding example. The **telnet** keyword is used instead of port 23.
access-list 101 deny udp 1.0.0.0 0.255.255.255 lt 1023 any	A packet with a source in network 1.0.0.0/8, using UDP with a source port less than (lt) 1023, with any destination IP address.

Extended IP ACL Configuration

Because extended ACLs can match so many different fields in the various headers in an IP packet, the command syntax cannot be easily summarized in a single generic command. However, the two commands in Table 26-5 summarize the syntax options as covered in this book.

Table 26-5 Extended IP Access List Configuration Commands

Command	Configuration Mode and Description
access-list *access-list-number* {**deny** | **permit**} *protocol source source-wildcard destination destination-wildcard* [**log** | **log-input**]	Global command for extended numbered **access lists**. Use a number between 100 and 199 or 2000 and 2699, inclusive
access-list *access-list-number* {**deny** | **permit**} {**tcp** | **udp**} *source source-wildcard* [*operator* [*port*]] *destination destination-wildcard* [*operator* [*port*]] [**established**] [**log**]	A version of the **access-list** command with parameters specific to TCP and/or UDP

The configuration process for extended ACLs mostly matches the same process used for standard ACLs. You must choose the location and direction in which to enable the ACL, particularly the direction, so that you can characterize whether certain addresses and ports will be either the source or destination. Configure the ACL using **access-list** commands, and when complete, then enable the ACL using the same **ip access-group** command used with standard ACLs. All these steps mirror what you do with standard ACLs; however, when configuring, keep the following differences in mind:

Key Topic

- Place extended ACLs as close as possible to the source of the packets that will be filtered. Filtering close to the source of the packets saves some bandwidth.

- Remember that all fields in one **access-list** command must match a packet for the packet to be considered to match that **access-list** statement.
- Use numbers of 100–199 and 2000–2699 on the **access-list** commands; no one number is inherently better than another.

Extended IP Access Lists: Example 1

This example focuses on understanding basic syntax. In this case, the ACL denies Bob access to all FTP servers on R1's Ethernet, and it denies Larry access to server1's web server. Figure 26-8 shows the network topology; Example 26-1 shows the configuration on R1.

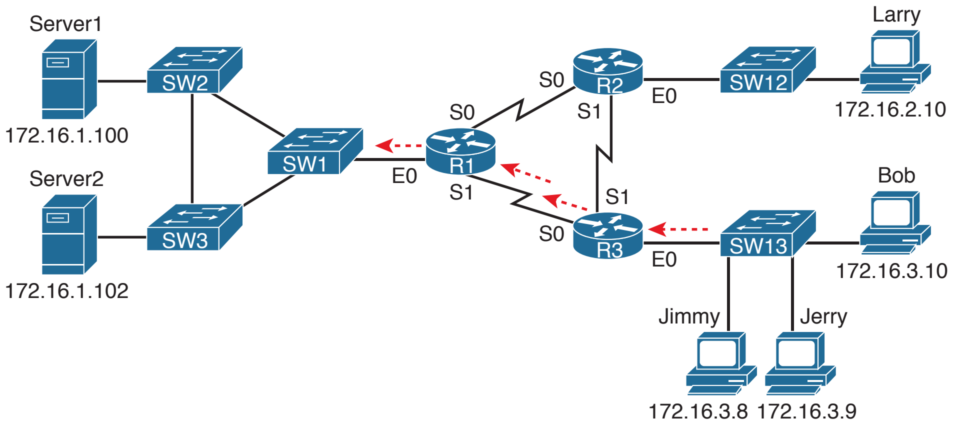


Figure 26-8 *Network Diagram for Extended Access List Example 1*

Example 26-1 *R1's Extended Access List: Example 1*

```
interface Serial0
 ip address 172.16.12.1 255.255.255.0
 ip access-group 101 in
!
interface Serial1
 ip address 172.16.13.1 255.255.255.0
 ip access-group 101 in
!
access-list 101 remark Stop Bob to FTP servers, and Larry to Server1 web
access-list 101 deny tcp host 172.16.3.10 172.16.1.0 0.0.0.255 eq ftp
access-list 101 deny tcp host 172.16.2.10 host 172.16.1.100 eq www
access-list 101 permit ip any any
```

The first ACL statement prevents Bob's access to FTP servers in subnet 172.16.1.0. The second statement prevents Larry's access to web services on Server1. The final statement permits all other traffic.

Focusing on the syntax for a moment, there are several new items to review. First, the access-list number for extended access lists falls in the range of 100 to 199 or 2000 to 2699. Following the **permit** or **deny** action, the *protocol* parameter defines whether you want to check for all IP packets or specific headers, such as TCP or UDP headers. When you check for TCP or UDP port numbers, you must specify the TCP or UDP protocol. Both FTP and web use TCP.

This example uses the **eq** parameter, meaning "equals," to check the destination port numbers for FTP control (keyword **ftp**) and HTTP traffic (keyword **www**). You can use the numeric values—or, for the more popular options, a more obvious text version is valid. (If you were to type **eq 80**, the config would show **eq www**.)

This example enables the ACL in two places on R1: inbound on each serial interface. These locations achieve the goal of the ACL. However, that initial placement was made to make the point that Cisco suggests that you locate them as close as possible to the source of the packet. Therefore, Example 26-2 achieves the same goal as Example 26-1 of stopping Bob's access to FTP servers at the main site, and it does so with an ACL on R3.

Example 26-2 *R3's Extended Access List Stopping Bob from Reaching FTP Servers Near R1*

```
interface Ethernet0
 ip address 172.16.3.1 255.255.255.0
 ip access-group 103 in

access-list 103 remark deny Bob to FTP servers in subnet 172.16.1.0/24
access-list 103 deny tcp host 172.16.3.10 172.16.1.0 0.0.0.255 eq ftp
access-list 103 permit ip any any
```

The new configuration on R3 meets the goals to filter Bob's traffic, while also meeting the overarching design goal of keeping the ACL close to the source of the packets. ACL 103 on R3 looks a lot like ACL 101 on R1 from Example 26-1, but this time, the ACL does not bother to check for the criteria to match Larry's traffic, because Larry's traffic will never enter R3's Ethernet 0 interface. ACL 103 filters Bob's FTP traffic to destinations in subnet 172.16.1.0/24, with all other traffic entering R3's E0 interface making it into the network.

Extended IP Access Lists: Example 2

Example 26-3, based on the network shown in Figure 26-9, shows another example of how to use extended IP access lists. This example uses the following criteria:

- Sam is not allowed access to the subnet of Bugs or Daffy.
- Hosts on the Seville Ethernet are not allowed access to hosts on the Yosemite Ethernet.
- All other combinations are allowed.

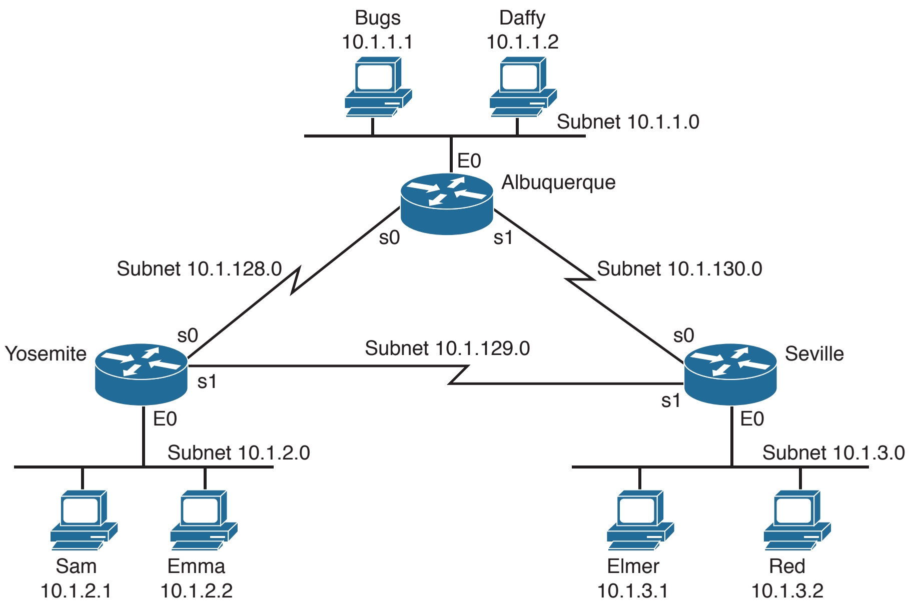


Figure 26-9 *Network Diagram for Extended Access List Example 2*

Example 26-3 *Yosemite Configuration for Extended Access List Example*

```
interface ethernet 0
 ip access-group 110 in
!
access-list 110 deny ip host 10.1.2.1 10.1.1.0 0.0.0.255
access-list 110 deny ip 10.1.2.0 0.0.0.255 10.1.3.0 0.0.0.255
access-list 110 permit ip any any
```

This configuration solves the problem with few statements while keeping to the Cisco design guideline of placing extended ACLs as close as possible to the source of the traffic. The ACL filters packets that enter Yosemite's E0 interface, which is the first router interface that packets sent by Sam enter. If the route between Yosemite and the other subnets changes over time, the ACL still applies. Also, the filtering mandated by the second requirement (to disallow Seville's LAN hosts from accessing Yosemite's) is met by the second **access-list** statement. Stopping packet flow from Yosemite's LAN subnet to Seville's LAN subnet stops effective communication between the two subnets. Alternatively, the opposite logic could have been configured at Seville.

Practice Building access-list Commands

Table 26-6 supplies a practice exercise to help you get comfortable with the syntax of the extended **access-list** command, particularly with choosing the correct matching logic. Your job: create a one-line extended ACL that matches the packets. The answers are in the section, "Answers to Earlier Practice Problems," later in this chapter. Note that if the criteria mentions a particular application protocol, for example, "web client," that means to specifically match for that application protocol.

Table 26-6 Building One-Line Extended ACLs: Practice

Problem	Criteria
1	From web client 10.1.1.1, sent to a web server in subnet 10.1.2.0/24.
2	From Telnet client 172.16.4.3/25, sent to a Telnet server in subnet 172.16.3.0/25. Match all hosts in the client's subnet as well.
3	ICMP messages from the subnet in which 192.168.7.200/26 resides to all hosts in the subnet where 192.168.7.14/29 resides.
4	From web server 10.2.3.4/23's subnet to clients in the same subnet as host 10.4.5.6/22.
5	From Telnet server 172.20.1.0/24's subnet, sent to any host in the same subnet as host 172.20.44.1/23.
6	From web client 192.168.99.99/28, sent to a web server in subnet 192.168.176.0/28. Match all hosts in the client's subnet as well.
7	ICMP messages from the subnet in which 10.55.66.77/25 resides to all hosts in the subnet where 10.66.55.44/26 resides.
8	Any and every IPv4 packet.

Named ACLs and ACL Editing

Now that you have a good understanding of the core concepts in IOS IP ACLs, this section examines a few enhancements to IOS support for ACLs: named ACLs and ACL editing with sequence numbers. Although both features are useful and important, neither adds any function as to what a router can and cannot filter. Instead, named ACLs and ACL sequence numbers make it easier to remember ACL names and edit existing ACLs when an ACL needs to change.

Named IP Access Lists

Named IP ACLs have many similarities with numbered IP ACLs. They can be used for filtering packets, plus for many other purposes. They can match the same fields as well: Standard numbered ACLs can match the same fields as a standard named ACL, and extended numbered ACLs can match the same fields as an extended named ACL.

Of course, there are differences between named and numbered ACLs. Named ACLs originally had three big differences compared to numbered ACLs:

Key Topic

- Using names instead of numbers to identify the ACL, making it easier to remember the reason for the ACL
- Using ACL subcommands, not global commands, to define the action and matching parameters
- Using ACL editing features that allow the CLI user to delete individual lines from the ACL and insert new lines

You can easily learn named ACL configuration by just converting numbered ACLs to use the equivalent named ACL configuration. Figure 26-10 shows just such a conversion, using a simple three-line standard ACL number 1. To create the three **permit** subcommands for the named ACL, you literally copy parts of the three numbered ACL commands, beginning with the **permit** keyword.

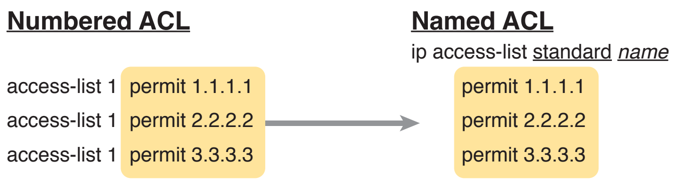

Numbered ACL
access-list 1 permit 1.1.1.1
access-list 1 permit 2.2.2.2
access-list 1 permit 3.3.3.3
Named ACL
ip access-list standard name
permit 1.1.1.1
permit 2.2.2.2
permit 3.3.3.3

Figure 26-10 *Named ACL Versus Numbered ACL Configuration*

The only truly new part of the named ACL configuration is the **ip access-list** global configuration command. This command defines whether an ACL is a standard or extended ACL, and defines the name. It also moves the user to ACL configuration mode, as shown in upcoming Example 26-4. Once in ACL configuration mode, you configure **permit**, **deny**, and **remark** commands that mirror the syntax of numbered ACL **access-list** commands. If you're configuring a standard named ACL, these commands match the syntax of standard numbered ACLs; if you're configuring extended named ACLs, they match the syntax of extended numbered ACLs.

Example 26-4 shows the configuration of a named extended ACL. Pay particular attention to the configuration mode prompts, which show ACL configuration mode.

Example 26-4 *Named Access List Configuration*

```
Router# configure terminal
Enter configuration commands, one per line.  End with Ctrl-Z.
Router(config)# ip access-list extended barney
Router(config-ext-nacl)# permit tcp host 10.1.1.2 eq www any
Router(config-ext-nacl)# deny udp host 10.1.1.1 10.1.2.0 0.0.0.255
Router(config-ext-nacl)# deny ip 10.1.3.0 0.0.0.255 10.1.2.0 0.0.0.255
Router(config-ext-nacl)# deny ip 10.1.2.0 0.0.0.255 10.2.3.0 0.0.0.255
Router(config-ext-nacl)# permit ip any any
Router(config-ext-nacl)# interface serial1
Router(config-if)# ip access-group barney out
Router(config-if)# ^Z
Router# show running-config
Building configuration...

Current configuration:

! lines omitted for brevity

interface serial 1
 ip access-group barney out
!
ip access-list extended barney
 permit tcp host 10.1.1.2 eq www any
 deny   udp host 10.1.1.1 10.1.2.0 0.0.0.255
 deny   ip 10.1.3.0 0.0.0.255 10.1.2.0 0.0.0.255
 deny   ip 10.1.2.0 0.0.0.255 10.2.3.0 0.0.0.255
 permit ip any any
```

Example 26-4 begins with the creation of an ACL named barney. The **ip access-list extended barney** command creates the ACL, naming it barney and placing the user in ACL configuration mode. This command also tells the IOS that barney is an extended ACL. Next, five different **permit** and **deny** statements define the matching logic and action to be taken upon a match. The **show running-config** command output lists the named ACL configuration before the single entry is deleted.

Named ACLs allow the user to delete and add new lines to the ACL from within ACL configuration mode. Example 26-5 shows how, with the **no deny ip . . .** command deleting a single entry from the ACL. Notice that the output of the **show access-list** command at the end of the example still lists the ACL, with four **permit** and **deny** commands instead of five.

Example 26-5 *Removing One Command from a Named ACL*

```
Router# configure terminal
Enter configuration commands, one per line.  End with Ctrl-Z.
Router(config)# ip access-list extended barney
Router(config-ext-nacl)# no deny ip 10.1.2.0 0.0.0.255 10.2.3.0 0.0.0.255
Router(config-ext-nacl)# ^Z
Router# show access-list

Extended IP access list barney
    10 permit tcp host 10.1.1.2 eq www any
    20 deny   udp host 10.1.1.1 10.1.2.0 0.0.0.255
    30 deny   ip 10.1.3.0 0.0.0.255 10.1.2.0 0.0.0.255
    50 permit ip any any
```

Editing ACLs Using Sequence Numbers

Numbered ACLs have existed in IOS since the early days of Cisco routers and IOS; however, for many years, through many IOS versions, the ability to edit a numbered IP ACL was poor. For example, to simply delete a line from the ACL, the user had to delete the entire ACL and then reconfigure it.

The ACL editing feature uses an ACL sequence number that is added to each ACL **permit** or **deny** statement, with the numbers representing the sequence of statements in the ACL. ACL sequence numbers provide the following features for both numbered and named ACLs:

Key Topic

New configuration style for numbered: Numbered ACLs use a configuration style like named ACLs, as well as the traditional style, for the same ACL; the new style is required to perform advanced ACL editing.

Deleting single lines: An individual ACL **permit** or **deny** statement can be deleted with a **no** *sequence-number* subcommand.

Inserting new lines: Newly added **permit** and **deny** commands can be configured with a sequence number before the **deny** or **permit** command, dictating the location of the statement within the ACL.

Automatic sequence numbering: IOS adds sequence numbers to commands as you configure them, even if you do not include the sequence numbers.

To take advantage of the ability to delete and insert lines in an ACL, both numbered and named ACLs must use the same overall configuration style and commands used for named ACLs. The only difference in syntax is whether a name or number is used. Example 26-6 shows the configuration of a standard numbered IP ACL, using this alternative configuration style. The example shows the power of the ACL sequence number for editing. In this example, the following occurs:

Step 1. Numbered ACL 24 is configured using this new-style configuration, with three **permit** commands.

Step 2. The **show ip access-lists** command shows the three permit commands with sequence numbers 10, 20, and 30.

Step 3. The engineer deletes only the second **permit** command using the **no 20** ACL subcommand, which simply refers to sequence number 20.

Step 4. The **show ip access-lists** command confirms that the ACL now has only two lines (sequence numbers 10 and 30).

Step 5. The engineer adds a new **deny** command to the beginning of the ACL, using the **5 deny 10.1.1.1** ACL subcommand.

Step 6. The **show ip access-lists** command again confirms the changes, this time listing three commands, sequence numbers 5, 10, and 30.

NOTE For this example, note that the user does not leave configuration mode, instead using the **do** command to tell IOS to issue the **show ip access-lists** EXEC command from configuration mode.

Example 26-6 *Editing ACLs Using Sequence Numbers*

```
! Step 1: The 3-line Standard Numbered IP ACL is configured.
R1# configure terminal
Enter configuration commands, one per line.  End with Ctrl-Z.
R1(config)# ip access-list standard 24
R1(config-std-nacl)# permit 10.1.1.0 0.0.0.255
R1(config-std-nacl)# permit 10.1.2.0 0.0.0.255
R1(config-std-nacl)# permit 10.1.3.0 0.0.0.255

! Step 2: Displaying the ACL's contents, without leaving configuration mode.
R1(config-std-nacl)# do show ip access-lists 24
Standard IP access list 24
    10 permit 10.1.1.0, wildcard bits 0.0.0.255
    20 permit 10.1.2.0, wildcard bits 0.0.0.255
    30 permit 10.1.3.0, wildcard bits 0.0.0.255

! Step 3: Still in ACL 24 configuration mode, the line with sequence number 20 is
  deleted.
R1(config-std-nacl)# no 20

! Step 4: Displaying the ACL's contents again, without leaving configuration mode.
! Note that line number 20 is no longer listed.
R1(config-std-nacl)#do show ip access-lists 24
Standard IP access list 24
    10 permit 10.1.1.0, wildcard bits 0.0.0.255
    30 permit 10.1.3.0, wildcard bits 0.0.0.255

! Step 5: Inserting a new first line in the ACL.
R1(config-std-nacl)# 5 deny 10.1.1.1

! Step 6: Displaying the ACL's contents one last time, with the new statement
!(sequence number 5) listed first.
R1(config-std-nacl)# do show ip access-lists 24
Standard IP access list 24
     5 deny   10.1.1.1
    10 permit 10.1.1.0, wildcard bits 0.0.0.255
    30 permit 10.1.3.0, wildcard bits 0.0.0.255
```

Note that although Example 26-6 uses a numbered ACL, named ACLs use the same process to edit (add and remove) entries.

Numbered ACL Configuration Versus Named ACL Configuration

As a brief aside about numbered ACLs, note that IOS actually allows two ways to configure numbered ACLs in the more recent versions of IOS. First, IOS supports the traditional method, using the **access-list** global commands shown earlier in Examples 26-1, 26-2, and 26-3. IOS also supports the numbered ACL configuration with commands just like named ACLs, as shown in Example 26-6.

Oddly, IOS always stores numbered ACLs with the original style of configuration, as global **access-list** commands, no matter which method is used to configure the ACL. Example 26-7 demonstrates these facts, picking up where Example 26-6 ended, with the following additional steps:

Step 7. The engineer lists the configuration (**show running-config**), which lists the old-style configuration commands—even though the ACL was created with the new-style commands.

Step 8. The engineer adds a new statement to the end of the ACL using the old-style **access-list 24 permit 10.1.4.0 0.0.0.255** global configuration command.

Step 9. The **show ip access-lists** command confirms that the old-style **access-list** command from the previous step followed the rule of being added only to the end of the ACL.

Step 10. The engineer displays the configuration to confirm that the parts of ACL 24 configured with both new-style commands and old-style commands are all listed in the same old-style ACL (**show running-config**).

Example 26-7 *Adding to and Displaying a Numbered ACL Configuration*

```
! Step 7: A configuration snippet for ACL 24.
R1# show running-config
```

```
! The only lines shown are the lines from ACL 24
access-list 24 deny   10.1.1.1
access-list 24 permit 10.1.1.0 0.0.0.255
access-list 24 permit 10.1.3.0 0.0.0.255

! Step 8: Adding a new access-list 24 global command
R1# configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
R1(config)# access-list 24 permit 10.1.4.0 0.0.0.255
R1(config)# ^Z

! Step 9: Displaying the ACL's contents again, with sequence numbers. Note that even
! the new statement has been automatically assigned a sequence number.
R1# show ip access-lists 24
Standard IP access list 24
    5 deny   10.1.1.1
    10 permit 10.1.1.0, wildcard bits 0.0.0.255
    30 permit 10.1.3.0, wildcard bits 0.0.0.255
    40 permit 10.1.4.0, wildcard bits 0.0.0.255

! Step 10: The numbered ACL configuration remains in old-style configuration commands.
R1# show running-config
! The only lines shown are the lines from ACL 24
access-list 24 deny   10.1.1.1
access-list 24 permit 10.1.1.0 0.0.0.255
access-list 24 permit 10.1.3.0 0.0.0.255
access-list 24 permit 10.1.4.0 0.0.0.255
```

ACL Implementation Considerations

ACLs can be a great tool to enhance the security of a network, but engineers should think about some broader issues before simply configuring an ACL to fix a problem. To help, Cisco makes the following general recommendations in the courses on which the CCNA R&S exams are based:

Key Topic

- Place extended ACLs as close as possible to the source of the packet. This strategy allows ACLs to discard the packets early.
- Place standard ACLs as close as possible to the destination of the packet. This strategy avoids the mistake with standard ACLs (which match the source IPv4 address only) of unintentionally discarding packets that did not need to be discarded.
- Place more specific statements early in the ACL.
- Disable an ACL from its interface (using the **no ip access-group** interface subcommand) before making changes to the ACL.

The first point deals with the concept of where to locate your ACLs. If you intend to filter a packet, filtering closer to the packet's source means that the packet takes up less bandwidth in the network, which seems to be more efficient—and it is. Therefore, Cisco suggests locating extended ACLs as close to the source as possible.

However, the second point seems to contradict the first point, at least for standard ACLs, to locate them close to the destination. Why? Well, because standard ACLs look only at the source IP address, they tend to filter more than you want filtered when placed close to the source. For example, imagine that Fred and Barney are separated by four routers. If you filter Barney's traffic sent to Fred on the first router, Barney can't reach any hosts near the other three routers. So, the Cisco courses make a blanket recommendation to locate standard ACLs closer to the destination to avoid filtering traffic you do not mean to filter.

For the third item in the list, by placing more specific matching parameters early in each list, you are less likely to make mistakes in the ACL. For example, imagine that the ACL first listed a command that permitted traffic going to 10.1.1.0/24, and the second command denied traffic going to host 10.1.1.1. Packets sent to host 10.1.1.1 would match the first command, and never match the more specific second command. Note that later IOS versions prevent this mistake during configuration in some cases, as shown later in this chapter in Example 26-11.

Finally, Cisco recommends that you disable the ACLs on the interfaces before you change the statements in the list. By doing so, you avoid issues with the ACL during an interim state. First, if you delete an entire ACL, and leave the IP ACL enabled on an interface with the **ip access-group** command, IOS does not filter any packets (that was not always the case in far earlier IOS versions)! As soon as you add one ACL command to that enabled ACL, however, IOS starts filtering packets based on that ACL. Those interim ACL configurations could cause problems.

For example, suppose you have ACL 101 enabled on S0/0/0 for output packets. You delete list 101 so that all packets are allowed through. Then, you enter a single **access-list 101** command. As soon as you press Enter, the list exists, and the router filters all packets exiting S0/0/0 based on the one-line list. If you want to enter a long ACL, you might temporarily filter packets you don't want to filter! Therefore, the better way is to disable the list from the interface, make the changes to the list, and then reenable it on the interface.

Troubleshooting with IPv4 ACLs

The use of IPv4 ACLs makes troubleshooting IPv4 routing more difficult. Any data plane troubleshooting process can include a catchall phrase to include checking for ACLs. A network can have all hosts working, DHCP settings correct, all LANs working, all router interfaces working, and all routers having learned all routes to all subnets—and ACLs can still filter packets. Although ACLs provide that important service of filtering some packets, ACLs can make the troubleshooting process that much more difficult.

This third of the three major sections of this chapter focuses on troubleshooting in the presence of IPv4 ACLs. It breaks the discussion into two parts. The first part gives advice about common problems you might see on the exam, and how to find those with **show** commands and some analysis. The second part then looks at how ACLs impact the **ping** command.

Analyzing ACL Behavior in a Network

ACLs cause some of the biggest challenges when troubleshooting problems in real networking jobs. The packets created by commands like **ping** and **traceroute** do not exactly match the fields in packets created by end users. The ACLs sometimes filter the **ping** and **traceroute** traffic, making the network engineer think some other kind of problems exists when no problems exist at all. Or, the problem with the end-user traffic really is caused by the ACL, but the ping and traceroute traffic works fine, because the ACL matches the end user traffic with a **deny** action but matches the ping and traceroute traffic with a **permit** action.

As a result, much of ACL troubleshooting requires thinking about ACL configuration versus the packets that flow in a network, rather that using a couple of IOS commands that identify the root cause of the problem. The **show** commands that help are those that give you the configuration of the ACL, and on what interfaces the ACL is enabled. You can also see statistics about which ACL statements have been matched. And using pings and traceroutes can help—as long as you remember that ACLs may apply different actions to those packets versus the end user traffic.

The following list phrases the ACL troubleshooting steps into a list for easier study. The list also expands on the idea of analyzing each ACL in Step 3. None of the ideas in the list are new compared to this chapter and the previous chapter, but it acts more as a summary of the common issues:

Key Topic

Step 1. Determine on which interfaces ACLs are enabled, and in which direction (**show running-config, show ip interfaces**).

Step 2. Find the configuration of each ACL (**show access-lists, show ip access-lists, show running-config**).

Step 3. Analyze the ACLs to predict which packets should match the ACL, focusing on the following points:

- **A. Misordered ACLs:** Look for misordered ACL statements. IOS uses first-match logic when searching an ACL.
- **B. Reversed source/destination addresses:** Analyze the router interface, the direction in which the ACL is enabled, compared to the location of the IP address ranges matched by the ACL statements. Make sure the source IP address field could match packets with that source IP address, rather than the destination, and vice versa for the destination IP address field.
- **C. Reversed source/destination ports:** For extended ACLs that reference UDP or TCP port numbers, continue to analyze the location and direction of the ACL versus the hosts, focusing on which host acts as the server using a well-known port. Ensure that the ACL statement matches the correct source or destination port depending on whether the server sent or will receive the packet.
- **D. Syntax:** Remember that extended ACL commands must use the **tcp** and **udp** keywords if the command needs to check the port numbers.
- **E. Syntax:** Note that ICMP packets do not use UDP or TCP; ICMP is considered to be another protocol matchable with the **icmp** keyword (instead of **tcp** or **udp**).
- **F. Explicit deny any:** Instead of using the implicit **deny any** at the end of each ACL, use an explicit configuration command to deny all traffic at the end of the ACL so that the **show** command counters increment when that action is taken.
- **G. Dangerous inbound ACLs:** Watch for inbound ACLs, especially those with deny all logic at the end of the ACL. These ACLs may discard incoming overhead protocols, like routing protocol messages.

H. **Standard ACL location:** Standard ACLs enabled close to the source of matched addresses can discard the packets as intended, but also discard packets that should be allowed through. Always pay close attention to the requirements of the ACL in these cases.

This chapter (and the previous) have already discussed the details of Step 3. The first two steps are important for Simlet questions in case you are not allowed to look at the configuration; you can use other **show** commands to determine all the relevant ACL configuration. The next few pages show some of the related commands and how they can uncover some of the issues described in the just-completed ACL troubleshooting checklist.

ACL Troubleshooting Commands

If you suspect ACLs are causing a problem, the first problem-isolation step is to find the location and direction of the ACLs. The fastest way to do this is to look at the output of the **show running-config** command and to look for **ip access-group** commands under each interface. However, in some cases, enable mode access may not be allowed, and **show** commands are required. Instead, use the **show ip interfaces** command to find which ACLs are enabled on which interfaces, as shown in Example 26-8.

Example 26-8 *Sample* **show ip interface** *Command*

```
R1> show ip interface s0/0/1
Serial0/0/1 is up, line protocol is up
  Internet address is 10.1.2.1/24
  Broadcast address is 255.255.255.255
  Address determined by setup command
  MTU is 1500 bytes
  Helper address is not set
  Directed broadcast forwarding is disabled
  Multicast reserved groups joined: 224.0.0.9
  Outgoing access list is not set
  Inbound  access list is 102
! roughly 26 more lines omitted for brevity
```

Note that the command output lists whether an ACL is enabled, in both directions, and which ACL it is. The example shows an abbreviated version of the **show ip interface S0/0/1** command, which lists messages for just this one interface. The **show ip interface** command would list the same messages for every interface in the router.

Step 2 of the ACL troubleshooting checklist then says that the contents of the ACL must be found. Again, the quickest way to look at the ACL is to use the **show running-config** command. If not available, the **show access-lists** and **show ip access-lists** commands list the same details shown in the configuration. These commands also list a useful counter that lists the number of packets that have matched each line in the ACL. Example 26-9 shows an example.

Example 26-9 **show ip access-lists** *Command Example*

```
R1# show ip access-lists
Extended IP access list 102
    10 permit ip 10.1.2.0 0.0.0.255 10.1.4.0 0.0.1.255 (15 matches)
```

The counter can be very useful for troubleshooting. If you can generate traffic that you think should match a particular line in an ACL, then you should see the matches increment on that counter. If you keep generating traffic that should match, but that line's counter never goes up, then those packets do not match that line in that ACL. Those packets could be matching an earlier line in the same ACL, or might not even be reaching that router (for any reason).

After the locations, directions, and configuration details of the various ACLs have been discovered in Steps 1 and 2, the hard part begins—analyzing what the ACL really does. For example, one of the most common tasks you will do is to look at the address fields and decide the range of addresses matched by that field. Remember, for an ACL that sits in a router configuration, you can easily find the address range. The low end of the range is the address (the first number), and the high end of the range is the sum of the address and wildcard mask. For instance, with ACL 102 in Example 26-9, which is obviously configured in some router, the ranges are as follows:

Source 10.1.2.0, wildcard 0.0.0.255: Matches from 10.1.2.0 through 10.1.2.255

Destination 10.1.4.0, wildcard 0.0.1.255: Matches from 10.1.4.0 through 10.1.5.255

The next few pages work through some analysis of a few of the items from Step 3 in the troubleshooting checklist.

Example Issue: Reversed Source/Destination IP Addresses

IOS cannot recognize a case in which you attempt to match the wrong addresses in the source or destination address field. So, be ready to analyze the enabled ACLs and their direction versus the location of different subnets in the network. Then ask yourself about the packets that drive that ACL: what could the source and destination addresses of those packets be? And does the ACL match the correct address ranges, or not?

For example, consider Figure 26-11, a figure that will be used in several troubleshooting examples in this chapter. The requirements for the next ACL follow the figure.

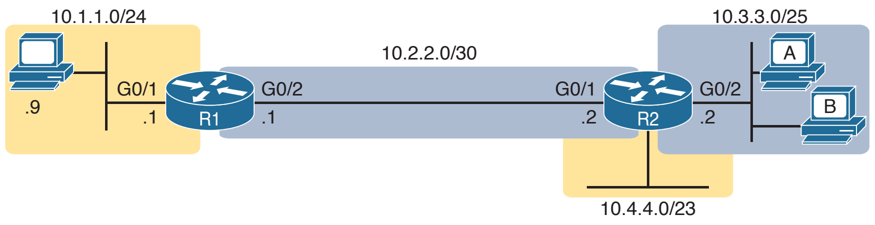


Figure 26-11 *Example Network Used in IPv4 ACL Troubleshooting Examples*

For this next ACL, the requirements ask that you allow and prevent various flows, as follows:

- Allow hosts in subnet 10.3.3.0/25 and subnet 10.1.1.0/24 to communicate
- Prevent hosts in subnet 10.4.4.0/23 and subnet 10.1.1.0/24 from communicating
- Allow all other communications between hosts in network 10.0.0.0
- Prevent all other communications

Example 26-10 shows the ACL used in this case on R2. At first glance, it meets all those requirements straight down the list.

Example 26-10 *Troubleshooting Example 2 per Step 3B: Source and Destination Mismatch*

```
R2# show ip access-lists
Standard IP access list Step3B
 10 permit 10.3.3.0 0.0.0.127
 20 deny 10.4.4.0 0.0.1.255
 30 permit 10.0.0.0 0.255.255.255 (12 matches)
R2#
R2# show ip interface G0/2 | include Inbound
  Inbound access list is Step3B
```

The problem in this case is that the ACL has been enabled on R2's G0/2 interface, inbound. Per the figure, packets coming from a source address in subnets 10.3.3.0/25 and 10.4.4.0/23 should be forwarded out R2's G0/2 interface, rather than coming in that interface. So, do not let the matching logic in the ACL that perfectly mirrors the requirements fool you; make sure and check the location of the ACL, direction, and the location of the IP addresses.

Note that Step 3C suggests a similar issue regarding matching well-known ports with TCP and UDP. The earlier section in this chapter titled "Matching TCP and UDP Port Numbers" has already discussed those ideas in plenty of detail. Just make sure to check where the server sits versus the location and direction of the ACL.

Steps 3D and 3E: Common Syntax Mistakes

Steps 3D and 3E describe a couple of common syntax mistakes. First, to match a TCP port in an ACL statement, you must use a **tcp** protocol keyword instead of **ip** or any other value. Otherwise, IOS rejects the command as having incorrect syntax. Same issue with trying to match UDP ports: a **udp** protocol keyword is required.

To match ICMP, IOS includes an **icmp** protocol keyword to use instead of **tcp** or **udp**. In fact, the main conceptual mistake is to think of ICMP as an application protocol that uses either UDP or TCP; it uses neither. To match all ICMP messages, for instance, use the **permit icmp any any** command in an extended named ACL.

Example Issue: Inbound ACL Filters Routing Protocol Packets

A router bypasses outbound ACL logic for packets the router itself generates. That might sound like common sense, but it is important to stop and think about that fact in context. A router can have an outgoing ACL, and that ACL can and will discard packets that the router receives in one interface and then tries to forward out some other interface. But if the router creates the packet, for instance, for a routing protocol message, the router bypasses the outbound ACL logic for that packet.

However, a router does not bypass inbound ACL logic. If an ACL has an inbound ACL enabled, and a packet arrives in that interface, the router checks the ACL. Any and all IPv4 packets are considered by the ACL—including important overhead packets like routing protocol updates.

For example, consider a seemingly good ACL on a router, like the Step3G ACL in Example 26-11. That ACL lists a couple of **permit** commands, and has an implicit deny any at the end of the list. At first, it looks like any other reasonable ACL.

Example 26-11 *Troubleshooting Example 2 per Step 3G: Filtering RIP by Accident*

```
R1# show ip access-lists
Standard IP access list Step3G
 10 permit host 10.4.4.1
 20 permit 10.3.3.0 0.0.0.127 (12 matches)
! using the implicit deny to match everything else
R1#
! On router R1:
R1# show ip interface G0/2 | include Inbound
  Inbound access list is Step3G
```

Now look at the location and direction (inbound on R1, on R1's G0/2) and consider that location versus the topology Figure 26-11 for a moment. None of those **permit** statements match the RIP updates sent by R2, sent out R2's G0/1 interface toward R1. RIP messages use UDP (well-known port 520), and R2's G0/1 interface is 10.2.2.2 per the figure. R1 would match incoming RIP messages with the implicit deny all at the end of the list. The symptoms in this case, assuming only that one ACL exists, would be that R1 would not learn routes from R2, but R2 could still learn RIP routes from R1.

Of the three routing protocols discussed in the ICND1 and ICND2 books, RIPv2 uses UDP as a transport, while OSPF and EIGRP do not even use a transport protocol. As a result, to match RIPv2 packets with an ACL, you need the **udp** keyword and you need to match well-known port 520. OSPF and EIGRP can be matched with special keywords as noted in Table 26-7. The table also list the addresses used by each protocol.

Table 26-7 Key Fields for Matching Routing Protocol Messages

Protocol	Source IP Address	Destination IP Addresses	ACL Protocol Keyword
RIPv2	Source interface	224.0.0.9	**udp** (port 520)
OSPF	Source interface	224.0.0.5, 224.0.0.6	**ospf**
EIGRP	Source interface	224.0.0.10	**eigrp**

Example 26-12 shows a sample ACL with three lines, one to match each routing protocol, just to show the syntax. Note that in this case, the ACL matches the address fields with the **any** keyword. You could include lines like these in any inbound ACL to ensure that routing protocol packets would be permitted.

Example 26-12 *Example ACL that Matches all RIPv2, OSPF, and EIGRP with a Permit*

```
R1# show ip access-lists
ip access-list extended RoutingProtocolExample
 10 permit udp any any eq 520
 20 permit ospf any any
 30 permit eigrp any any
 remark a complete ACL would also need more statements here
R1#
```

ACL Interactions with Router-Generated Packets

Routers bypass outbound ACL logic for packets generated by that same router. This logic helps avoid cases in which a router discards its own overhead traffic. This logic applies to packets that a router creates for overhead processes like routing protocols, as well as for commands, like **ping** and **traceroute**. This section adds a few perspectives about how ACLs impact troubleshooting, and how this exception to outbound ACL logic applies, particularly commands used from the router CLI.

Local ACLs and a Ping from a Router

For the first scenario, think about a ping command issued by a router. The command generates packets, and the router sends those packets (holding the ICMP echo request messages) out one of the router interfaces, and typically some ICMP echo reply messages are received back. As it turns out, not all ACLs will attempt to filter those packets.

As a backdrop to discuss what happens, Figure 26-12 illustrates a simple network topology with two routers connected to a serial link. Note that in this figure four IP ACLs exist, named A, B, C, and D, as noted by the thick arrows in the drawing. That is, ACL A is an outbound ACL on R1's S0/0/0, ACL B is an inbound ACL on R2's S0/0/1, and so on.

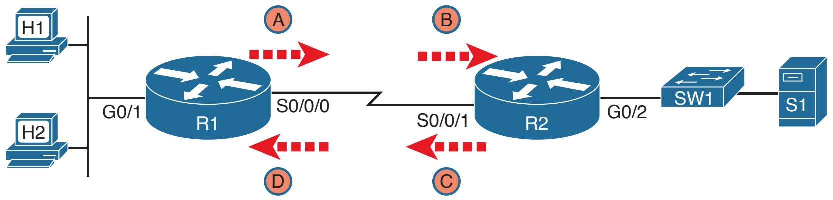


Figure 26-12 *Sample Network with IP ACLs in Four Locations*

As an example, consider a **ping** command issued from R1's CLI (after a user connects to R1's CLI using SSH). The **ping** command pings server S1's IP address. The IPv4 packets with the ICMP messages flow from R1 to S1 and back again. Which of those four ACLs could possibly filter the ICMP Echo Request toward S1, and the ICMP Echo Reply back toward R1?

Routers bypass their own outbound ACLs for packets generated by the router, as shown in Figure 26-13. Even though ACL A exists as an outgoing ACL on Router R1, R1 bypasses its own outgoing ACL logic of ACL A for the ICMP Echo Requests generated by R1.

Key Topic

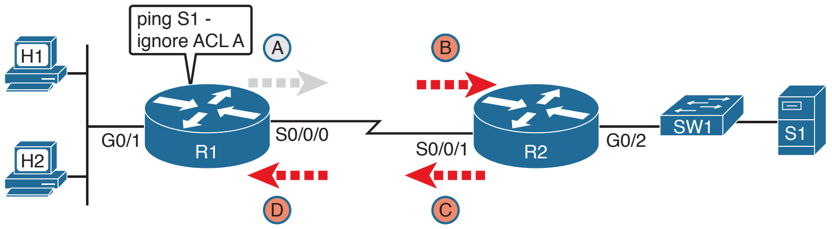


Figure 26-13 *R1 Ignores Outgoing ACL for Packets Created by Its Own* **ping** *Command*

Router Self-Ping of a Serial Interface IPv4 Address

The previous example uses a router's ping command when pinging a host. However, network engineers often need to ping router IP addresses, including using a self-ping. The term *self-ping* refers to a ping of a device's own IPv4 address. And for point-to-point serial links, a self-ping actually sends packets over the serial link, which causes some interested effects with ACLs.

When a user issues a self-ping for that local router's serial IP address, the router actually sends the ICMP echo request out the link to the other router. The neighboring router then receives the packet and routes the packet with the ICMP echo request back to the original router. Figure 26-14 shows an example of a self-ping (**ping 172.16.4.1**) of Router R1's own IP address on a point-to-point serial link, with the ICMP echo request out the link to Router R2. At Step 2, R2 treats it like any other packet not destined for one of R2's own IPv4 addresses: R2 routes the packet. Where? Right back to Router R1, as shown in the figure.

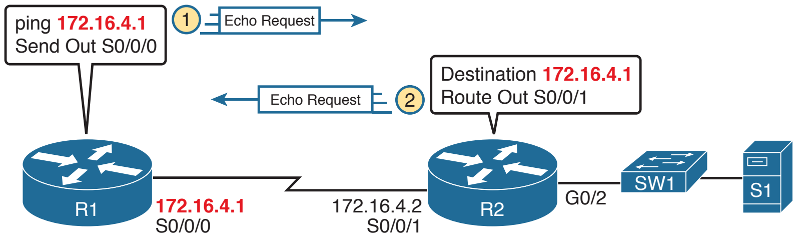


Figure 26-14 *The First Steps in a Self-Ping on R1, for R1's S0/0/0 IP Address*

Now think about those four ACLs in the earlier figures compared to Figure 26-14. R1 generates the ICMP echo request, so R1 bypasses outbound ACL A. ACLs B, C, and D could filter the packet. Note that the packet sent by R2 back to R1 is not generated by R2 in this case; R2 is just routing R1's original packet back to R1.

A self-ping of a serial interface actually tests many parts of a point-to-point serial link, as follows:

- The link must work at Layers 1, 2, and 3. Specifically, both routers must have a working (up/up) serial interface, with correct IPv4 addresses configured.
- ACLs B, C, and D must permit the ICMP echo request and reply packets.

So, when troubleshooting, if you choose to use self-pings and they fail, but the serial interfaces are in an up/up state, do not forget to check to see whether the ACLs have filtered the Internet Control Management Protocol (ICMP) traffic.

Router Self-Ping of an Ethernet Interface IPv4 Address

A self-ping of a router's own Ethernet interface IP address works a little like a self-ping of a router's serial IP address, but with a couple of twists:

- Like with serial interface, the local router interface must be working (in an up/up state); otherwise, the ping fails.
- Unlike serial interfaces, the router does not forward the ICMP messages physically out the interface, so security features on neighboring switches (like port security) or routers (like ACLs) cannot possibly filter the messages used by the **ping** command.
- Like serial interfaces, an incoming IP ACL on the local router does process the router self-ping of an Ethernet-based IP address.

Figure 26-15 walks through an example. In this case, R2 issues a **ping 172.16.2.2** command to ping its own G0/2 IP address. Just like with a self-ping on serial links, R2 creates the

ICMP echo request. However, R2 basically processes the ping down its own TCP/IP stack and back up again, with the ICMP echo never leaving the router's Ethernet interface. R2 does check the Ethernet interface status, showing a failure if the interface is not up/up. R2 does not apply outbound ACL logic to the packet, because R2 created the packet, but R2 will apply inbound ACL logic to the packet, as if the packet had been physically received on the interface.

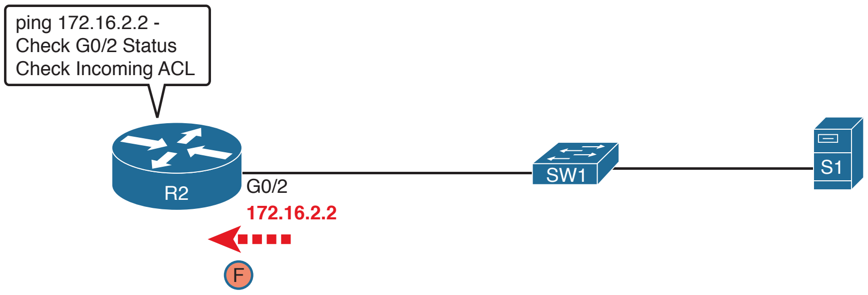


Figure 26-15 *Self-Ping of a Router's Ethernet Address*

Chapter Review

One key to doing well on the exams is to perform repetitive spaced review sessions. Review this chapter's material using either the tools in the book, DVD, or interactive tools for the same material found on the book's companion website. Refer to the "Your Study Plan" element for more details. Table 26-8 outlines the key review elements and where you can find them. To better track your study progress, record when you completed these activities in the second column.

Table 26-8 Chapter Review Tracking

Review Element	Review Date(s)	Resource Used
Review key topics		Book, DVD/website
Review key terms		Book, DVD/website
Repeat DIKTA questions		Book, PCPT
Review memory tables		Book, DVD/website
Review command tables		Book

Review All the Key Topics

Key Topic

Table 26-9 Key Topics for Chapter 26

Key Topic Element	Description	Page Number
Figure 26-3	Syntax and notes about the three required matching fields in the extended ACL **access-list** command	617
Paragraph	Summary of extended ACL logic that all parameters must match in a single **access-list** statement for a match to occur	618

Key Topic Element	Description	Page Number
Figure 26-4	Drawing of the IP header followed by a TCP header	618
Figure 26-5	Syntax and notes about matching TCP and UDP ports with extended ACL **access-list** commands	619
Figure 26-7	Logic and syntax to match TCP source ports	620
List	Guidelines for using extended numbered IP ACLs	621
List	Differences between named and numbered ACLs when named ACLs introduced	625
List	Features enabled by IOS 12.3 ACL sequence numbers	627
List	ACL implementation recommendations	630
Checklist	ACL troubleshooting checklist	632
Figure 26-13	Example of a router bypassing its outbound ACL logic for packets the router generates	637

Key Terms You Should Know

extended access list, named access list

Command References

Tables 26-10 and 26-11 list configuration and verification commands used in this chapter. As an easy review exercise, cover the left column in a table, read the right column, and try to recall the command without looking. Then repeat the exercise, covering the right column, and try to recall what the command does.

Table 26-10 Chapter 26 ACL Configuration Command Reference

Command	Description
access-list *access-list-number* {**deny** | **permit**} *protocol source source-wildcard destination destination-wildcard* [**log**]	Global command for extended numbered **access lists**. Use a number between 100 and 199 or 2000 and 2699, inclusive.
access-list *access-list-number* {**deny** | **permit**} **tcp** *source source-wildcard* [*operator* [*port*]] *destination destination-wildcard* [*operator* [*port*]] [**log**]	A version of the **access-list** command with TCP-specific parameters.
access-list *access-list-number* **remark** *text*	Defines a remark that helps you remember what the ACL is supposed to do.
ip access-group {*number* | *name* [**in** | **out**]}	Interface subcommand to enable access lists.
access-class *number* | *name* [**in** | **out**]	Line subcommand to enable either standard or extended access lists on vty lines.
ip access-list {**standard** | **extended**} *name*	Global command to configure a named standard or extended ACL and enter ACL configuration mode.
{**deny** | **permit**} *source* [*source wildcard*] [**log**]	ACL mode subcommand to configure the matching details and action for a standard named ACL.

Command	Description
{**deny** | **permit**} *protocol source source-wildcard destination destination-wildcard* [**log**]	ACL mode subcommand to configure the matching details and action for an extended named ACL.
{**deny** | **permit**} **tcp** *source source-wildcard* [*operator* [*port*]] *destination destination-wildcard* [*operator* [*port*]] [**log**]	ACL mode subcommand to configure the matching details and action for a named ACL that matches TCP segments.
remark *text*	ACL mode subcommand to configure a description of a named ACL.

Table 26-11 Chapter 26 EXEC Command Reference

Command	Description
show ip *interface* [*type number*]	Includes a reference to the access lists enabled on the interface
show access-lists [*access-list-number* | *access-list-name*]	Shows details of configured access lists for all protocols
show ip access-lists [*access-list-number* | *access-list-name*]	Shows IP access lists

Answers to Earlier Practice Problems

Table 26-12 lists the answers to the practice problems listed in Table 26-6. Note that for any question that references a client, you might have chosen to match port numbers greater than 1023. The answers in this table mostly ignore that option, but just to show one sample, the answer to the first problem lists one with a reference to client ports greater than 1023 and one without. The remaining answers simply omit this part of the logic.

Table 26-12 Building One-Line Extended ACLs: Answers

	Criteria
1	**access-list 101 permit tcp host 10.1.1.1 10.1.2.0 0.0.0.255 eq www** or **access-list 101 permit tcp host 10.1.1.1 gt 1023 10.1.2.0 0.0.0.255 eq www**
2	**access-list 102 permit tcp 172.16.4.0 0.0.0.127 172.16.3.0 0.0.0.127 eq telnet**
3	**access-list 103 permit icmp 192.168.7.192 0.0.0.63 192.168.7.8 0.0.0.7**
4	**access-list 104 permit tcp 10.2.2.0 0.0.1.255 eq www 10.4.4.0 0.0.3.255**
5	**access-list 105 permit tcp 172.20.1.0 0.0.0.255 eq 23 172.20.44.0 0.0.1.255**
6	**access-list 106 permit tcp 192.168.99.96 0.0.0.15 192.168.176.0 0.0.0.15 eq www**
7	**access-list 107 permit icmp 10.55.66.0 0.0.0.127 10.66.55.0 0.0.0.63**
8	**access-list 108 permit ip any any**

5. A single-line ACL has been added to a router configuration using the command **ip access-list 1 permit 172.16.4.0 0.0.1.255**. The configuration also includes the **ip access-class 1 in** command in VTY configuration mode. Which answer accurately describes how the router uses ACL 1?

a. Hosts in subnet 172.16.4.0/23 alone can telnet into the router.

b. CLI users cannot telnet from the router to hosts in subnet 172.16.4.0/23 alone.

c. Hosts in subnet 172.16.4.0/23 alone can log in but cannot reach enable mode of the router.

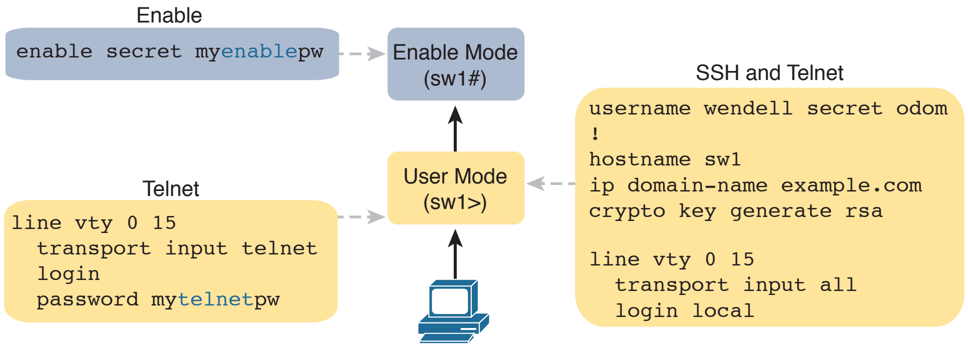
d. The router will only forward packets with source addresses in subnet 172.16.4.0/23.

Foundation Topics

Securing IOS Passwords

The ultimate way to protect passwords in Cisco IOS devices is to not store passwords in IOS devices. That is, for any functions that can use an external Authentication, Authorization, and Accounting (AAA) server, use it. However, it is common to store some passwords in a router or switch configuration, and this first section of the chapter discusses some of the ways to protect those passwords.

As a brief review, Figure 34-1 summarizes some typical login security configuration on a router or switch. On the lower left, you see Telnet support configured, with the use of a password only (no username required). On the right, the configuration adds support for login with both username and password, supporting both Telnet and SSH users. The upper left shows the one command required to define an enable password in a secure manner.



```
enable secret myenablepw
```

```
line vty 0 15
  transport input telnet
  login
  password mytelnetpw
```

```
username wendell secret odom
!
hostname sw1
ip domain-name example.com
crypto key generate rsa

line vty 0 15
  transport input all
  login local
```

Figure 34-1 *Example Login Security Configuration*

The rest of this first section of the chapter discusses how to make these passwords secure. In particular, this section looks at ways to avoid keeping clear-text passwords in the configuration and storing the passwords in ways that make it difficult for attackers to learn the password.

Encrypting Older IOS Passwords with service password-encryption

Some older-style IOS passwords create a security exposure because the passwords exist in the configuration file as clear-text. These clear-text passwords might be seen in printed versions of the configuration files, in a backup copy of the configuration file stored on a server, or as displayed on a network engineer's display.

Cisco attempted to solve this clear-text problem by adding a command to encrypt those passwords: the **service password-encryption** global configuration command. This command encrypts passwords that are normally held as clear text, specifically the passwords for these commands:

Key Topic

password *password* (console or vty mode)

username *name* **password** *password* (global)

enable *name* **password** *password* (global)

To see how it works, Example 34-1 shows how the **service password-encryption** command encrypts the clear-text console password. The example uses the **show running-config | section line con 0** command both before and after the encryption; this command lists only the section of the configuration about the console.

Example 34-1 *Encryption and the* **service password-encryption** *Command*

```
Switch3# show running-config | section line con 0
line con 0
 password cisco
 login

Switch3# configure terminal
Enter configuration commands, one per line.   End with CNTL/Z.
Switch3(config)# service password-encryption
Switch3(config)# ^Z

Switch3# show running-config | section line con 0
line con 0
 password 7 070C285F4D06
 login
```

A close examination of the before and after **show running-config** command output reveals both the obvious effect and a new concept. The encryption process now hides the original clear-text password. Also, IOS needs a way to signal that the value in the **password** command lists an encrypted password, rather than the clear text. IOS adds the encryption or encoding type of "7" to the command, which specifically refers to passwords encrypted with the **service password-encryption** command. (IOS considers the clear-text passwords to be type 0; some commands list the 0, and some do not.)

Answers to the "Do I Know This Already?" quiz:

1 B **2** A **3** B **4** B **5** A

While the **service password-encryption** global command encrypts passwords, the **no service password-encryption** global command does not immediately decrypt the passwords back to their clear-text state. Instead, the process works as shown in Figure 34-2. Basically, after you enter the **no service password-encryption** command, the passwords remain encrypted until you change a password.

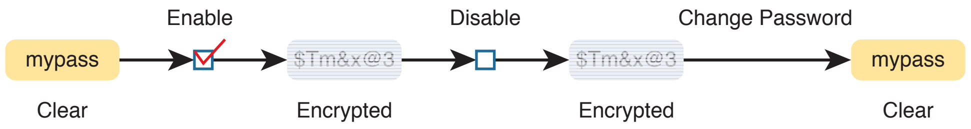


Figure 34-2 *Encryption is Immediate, Decryption Awaits Next Password Change*

Unfortunately, the **service password-encryption** command does not protect the passwords very well. Armed with the encrypted value, you can search the Internet and find sites with tools to decrypt these passwords. In fact, you can take the encrypted password from this example, plug it into one of these sites, and it decrypts to "cisco." So, the **service password-encryption** command will slow down the curious, but it will not stop a knowledgeable attacker.

Encoding the Enable Passwords with Hashes

In the earliest days of IOS, Cisco used the **enable password** *password* global command to define the password that users had to use to reach enable mode (after using the **enable** EXEC command). However, as just noted, the **enable password** *password* command stored the password as clear text, or was encrypted in a way that was easily decrypted.

Cisco solved the problem of only weak ways to store the password of the **enable password** *password* global command by making a more secure replacement: the **enable secret** *password* global command. However, both these commands exist in IOS even today. The next few pages look at these two commands from a couple of angles, including interactions between these two commands, why the **enable secret** command is more secure, along with a note about some advancements in how IOS secures the **enable secret** password.

Interactions Between Enable Password and Enable Secret

First, for real life: use the **enable secret** *password* global command, and ignore the **enable password** *password* global command. That has been true for around 20 years.

However, to be complete, Cisco has never removed the much weaker **enable password** command from IOS. So, on a single switch (or router), you can configure one or the other, both, or neither. What then does the switch expect us to type as the password to reach enable mode? It boils down to these rules:

Key Topic

Both commands configured: Users must use the password in the **enable secret** *password* command (and ignore the **enable password** *password* command).

Only one command configured: Use the password in that one command.

Neither command configured (default): Console users move directly to enable mode without a password prompt; Telnet and SSH users are rejected with no option to supply an enable password.

Making the Enable Secret Truly Secret with a Hash

Cisco's **enable secret** command protects the password value by never even storing the clear-text password in the configuration. However, that one sentence may cause you a bit of confusion: If the router or switch does not remember the clear-text password, how can the switch know that the user typed the right password after using the **enable** command? This section works through a few basics to show you how and appreciate why the password's value is secret.

First, by default, IOS uses a hash function called Message Digest 5 (MD5) to store an alternative value in the configuration, rather than the clear-text password. Think of MD5 as a rather complex mathematical formula. In addition, this formula is chosen so that even if you know the exact result of the formula—that is, the result after feeding the clear-text password through the formula as input—it is computationally difficult to compute the original clear-text password. Figure 34-3 shows the main ideas:

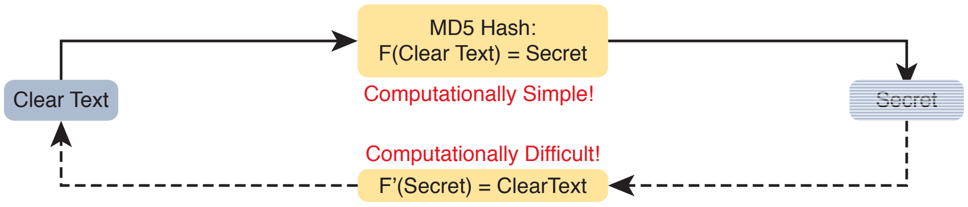


Figure 34-3 *One-Way Nature of MD5 Hash to Create Secret*

NOTE "Computationally difficult" is almost a code phrase, meaning that the designers of the function hope that no one is willing to take the time to compute the original clear text.

So, if the original clear-text password cannot be re-created, how can a switch or router use it to compare to the clear-text password typed by the user? The answer depends on another fact about these security hashes like MD5: Each clear-text input results in a unique result from the math formula.

The **enable secret fred** command generates an MD5 hash. If a user types "fred" when trying to enter enable mode, IOS will run MD5 against that value and get the same MD5 hash as is listed in the **enable secret** command, so IOS allows the user to access enable mode. If the user typed any other value besides "fred", IOS would compute a different MD5 hash than the value stored with the **enable secret** command, and IOS would reject that user's attempt to reach enable mode.

Knowing that fact, the switch can make a comparison when a user types a password after using the **enable** EXEC command as follows:

Key Topic

Step 1. IOS computes the MD5 hash of the password in the **enable secret** command and stores the hash of the password in the configuration.

Step 2. When the user types the **enable** command to reach enable mode, a password that needs to be checked against that configuration command, IOS hashes the clear-text password as typed by the user.

Step 3. IOS compares the two hashed values: If they are the same, the user-typed password must be the same as the configured password.

As a result, IOS can store the hash of the password but never store the clear-text password; however, it can still determine whether the user typed the same password.

Switches and routers already use the logic described here, but you can see the evidence by looking at the switch configuration. Example 34-2 shows the creation of the **enable secret** command, with a few related details. This example shows the stored (hashed) value as revealed in the **show running-configuration** command output. That output also shows that IOS changed the **enable secret fred** command to list the encryption type 5 (which means the listed password is actually an MD5 hash of the clear-text password). The gobbledygook long text string is the hash, preventing others from reading the password.

Example 34-2 *Cisco IOS Encoding Password "cisco" as Type 5 (MD5)*

```
Switch3(config)# enable secret fred
Switch3(config)# ^Z
Switch3# show running-config | include enable secret

enable secret 5 $1$ZGMA$e8cmvkz4UjiJhVp7.maLE1

Switch3# configure terminal
Enter configuration commands, one per line.   End with CNTL/Z.
Switch3(config)# no enable secret
Switch3(config)# ^Z
```

The end of the example also shows an important side point about deleting the **enable secret** password: After you are in enable mode, you can delete the enable secret password using the **no enable secret** command, without even having to enter the password value. You can also overwrite the old password by just repeating the **enable secret** command. But you cannot view the original clear-text password.

NOTE Example 34-2 shows another shortcut illustrating how to work through long **show** command output, this time using the pipe to the **include** command. The **| include enable secret** part of the command processes the output from **show running-config** to include only the lines with the case-sensitive text "enable secret."

Improved Hashes for Cisco's Enable Secret

The use of any hash function to encode passwords relies on several key features of the particular hash function. In particular, every possible input value must result in a single hashed value, so that when users type a password, only one password value matches each hashed value. Also, the hash algorithm must be computationally difficult (in other words, a pain in the neck) to compute the clear-text password based on the hashed value to discourage attackers.

As of the publication of this book in 2016, the MD5 hash algorithm has been around about 25 years. Over those years, computers have gotten much faster, and researchers have found creative ways to attack the MD5 algorithm, making MD5 less challenging to crack. That is, someone who saw your running configuration would have an easier time re-creating your clear-text secret passwords.

These facts are not meant to say that MD5 was bad, but like many cryptographic functions before MD5, progress was made, and new functions were needed. (In fact, the **enable secret** command's use of MD5 as its only hash algorithm spanned over 15 years.) Cisco has begun to improve the way that it encodes secret passwords in IOS. At press time, Cisco has added two newer security hashes for passwords to router IOS images, as noted in Figure 34-4.

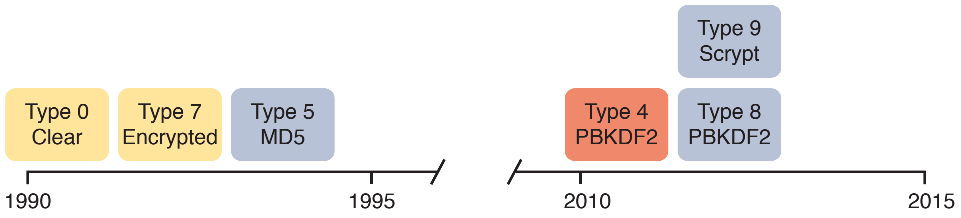


Figure 34-4 *Timeline of Encryptions/Hashes of Cisco IOS Passwords*

IOS now supports two alternative algorithm types in the more recent router IOS images. Both use an SHA-256 hash instead of MD5, but with some differences in the particulars of how each algorithm uses SHA-256. Table 34-2 shows the configuration of all three algorithm types on the **enable secret** command.

Table 34-2 Commands and Encoding Types for the **enable secret** Command

Command	Type	Algorithm
enable [algorithm-type md5] secret *password*	5	MD5
enable algorithm-type sha-256 secret *password*	8	SHA-256
enable algorithm-type scrypt secret *password*	9	SHA-256

34

Example 34-3 shows the **enable secret** being changed from MD5 to the scrypt algorithm. Of note, the example shows that only one **enable secret** command should exist between those three commands in Table 34-2. Basically, if you configure another **enable secret** command with a different algorithm type, that command replaces any existing **enable secret** command.

Example 34-3 *Cisco IOS Encoding Password "mypass1" as Type 9 (SHA-256)*

```
R1# show running-config | include enable
enable secret 5 $1$ZSYj$725dBZmLUJ0nx8gFPTtTv0
R1# configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
R1(config)# enable algorithm-type scrypt secret mypass1
R1(config)# ^Z
R1#
R1# show running-config | include enable
enable secret 9 $9$II/EeKiRW91uxE$fwYuOE5EHoii16AWv2wSywkLJ/KNeGj8uK/24B0TVU6
R1#
```

Following the process shown in the example, the first command confirms that the current **enable secret** command uses encoding type 5, meaning it uses MD5. Second, the user configures the password using algorithm type scrypt. The last command confirms that only one **enable secret** command exists in the configuration, now with encoding type 9.

Once you copy the IOS file into a local IOS file system on the router, you must **reload** the router to start using the new IOS. The next topic looks at the entire IOS boot process, including how to make a router start using the new version of IOS.

The Cisco IOS Software Boot Sequence

Cisco routers perform the same types of tasks that a typical computer performs when you power it on or reboot (reload) it. However, most end-user computers have a single instance of the OS installed, so the computer does not have to choose which OS to load. In contrast, a router can have multiple IOS images available both in flash memory and on external servers, so the router needs a process to pick which IOS image to load into RAM and use. This section examines the entire boot process, with extra emphasis on the options that impact a router's choice of what IOS image to load.

NOTE Routers can load IOS or a special-purpose OS called ROMMON. ROMMON is used for special purposes like password recovery. ROMMON can be used to send and receive IP packets to load a new IOS, but it does not route packets. A third very old special-purpose OS, called RXBOOT, is no longer included in this book because it applies only to very old routers models.

When a router first powers on, it follows these four steps:

Key Topic

Step 1. The router performs a power-on self-test (POST) process to discover the hardware components and verify that all components work properly.

Step 2. The router copies a bootstrap program from ROM into RAM and runs the bootstrap program.

Step 3. The bootstrap program decides which IOS image (or the ROMMON OS) to load into RAM, and then the bootstrap program loads the OS. After loading the chosen OS image, the bootstrap program hands over control of the router hardware to the newly loaded OS.

Step 4. If the bootstrap program happened to load IOS, once IOS is running, it finds the startup-config file and loads it into RAM as the running-config.

All routers attempt all four steps each time the router is powered on or reloaded. The first two steps do not have any options to choose; either both of these steps succeed or the initialization fails. If it fails, you might need to call the Cisco Technical Assistance Center (TAC) for support. However, Steps 3 and 4 have several configurable options that tell the router what to do next, as noted in Figure 35-3.

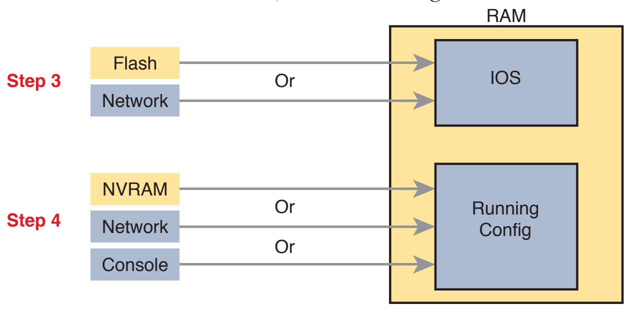


Figure 35-3 *Loading IOS and Initial Configuration*

As you can see, the router has options at both Steps 3 and 4 in the figure. However, at Step 4, routers almost always load the configuration from NVRAM (the startup-config file), when it exists. There is no real advantage to storing the initial configuration anywhere else except NVRAM, so this chapter does not look further into the options of Step 4. But there are reasonable motivations for keeping IOS images in flash and on servers in the network, so the rest of this section examines Step 3 in more detail.

The Configuration Register

A router's configuration register has an impact on a router's choice of which OS to load.

Routers use a *configuration register* to find some configuration settings at boot time, before the router has loaded IOS and read the startup-config file. The 16 bits (4 hex digits) in the configuration register set a variety of different parameters. For example, the console runs at a speed of 9600 bps by default, but that console speed is based on the default settings of a couple of bits in the configuration register. By changing specific bits in the configuration register, the next time the router boots, you can change the speed of the console line.

You can set the configuration register value with the **config-register** global configuration command. Engineers set the configuration register to different values for many reasons, but the most common are to help tell the router what IOS image to load, as explained in the next few pages, and in the password recovery process. For example, the global configuration command **config-register 0x2100** sets the value to hexadecimal 2100, which causes the router to load the ROMMON OS rather than IOS the next time the router is reloaded.

Interestingly, Cisco routers automatically save the new configuration register value when you press **Enter** at the end of the **config-register** command; you do not need to use the **copy running-config startup-config** command after changing the configuration register. However, the configuration register's new value has no effect until the next time the router is reloaded.

NOTE On most Cisco routers, the default configuration register setting is hexadecimal 2102, which leaves the console speed at 9600 bps and tells the router to load an IOS image.

How a Router Chooses Which OS to Load

A router chooses the OS to load based on two factors:

- The last hex digit in the configuration register (called the *boot field*)
- Any **boot system** global configuration commands in the startup-config file

The boot field, the fourth hex digit in the configuration register, tells the router the initial instructions about what OS to try to load. The router looks at the boot field's value when the router is powered on or when reloaded. The boot field's value then tells the router how to proceed with choosing which OS to load.

NOTE Cisco represents hexadecimal values by preceding the hex digits with 0x; for example, 0xA would mean a single hex digit A.

The process to choose which OS to load on modern Cisco routers happens as follows:

Key Topic

1. If boot field = 0, use the ROMMON OS.
2. If boot field = 1, load the first IOS file found in flash memory.
3. If boot field = 2-F:
   A. Try each **boot system** command in the startup-config file, in order, until one works.
   B. If none of the **boot system** commands work, load the first IOS file found in flash memory.
4. If all other attempts fail, load ROMMON, from which you can perform further steps to recover by copying a new IOS image into flash.

NOTE The actual step numbers are not important; the list is just numbered for easier reference.

The first two steps are pretty straightforward, but Step 3 then tells the router to look to the second major method to tell the router which IOS to load: the **boot system** global configuration command. This command can be configured multiple times on one router, with each new **boot system** command being added to the end of a list of **boot system** commands. Each command can point to different files in flash memory, and filenames and IP addresses of servers, telling the router where to look for an IOS image to load. The router tries to load the IOS images in the order of the configured **boot system** commands.

Both Step 2 and Step 3B refer to a concept of the "first" IOS file, a concept that needs a little more explanation. Routers number the files stored in flash memory, with each new file usually getting a higher and higher number. When a router tries Step 2 or Step 3B from the preceding list, the router looks in flash memory, starting with file number 1, and then file number 2, and so on, until it finds the lowest numbered file that happens to be an IOS image. The router then loads that file.

Interestingly, most routers end up using Step 3B to find their IOS image. From the factory, Cisco routers do not have any **boot system** commands configured; in fact, they do not have any configuration in the startup-config file at all. Cisco loads flash memory with a single IOS when it builds and tests the router, and the configuration register value is set to 0x2102, meaning a boot field of 0x2. With all these settings, the process tries Step 3 (because boot = 2), finds no **boot system** commands (because the startup-config is empty), and then looks for the first file in flash memory at Step 3B.

NOTE Routers do not search all flash file systems for an IOS image. The details vary depending on the router model, but routers consider one flash file system to be the default IOS file system to look for IOS images.

Figure 35-4 summarizes the key concepts behind how a router chooses the OS to load.

Key Topic

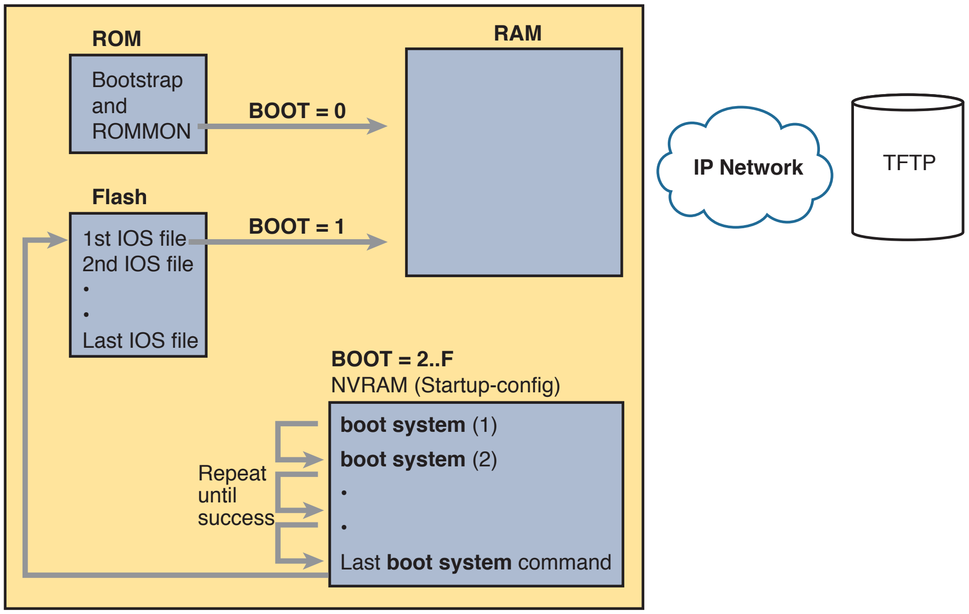


Figure 35-4 *Choices for Choosing the OS at Boot Time: Modern Cisco Router*

The **boot system** commands need to refer to the exact file that the router should load. Table 35-3 shows several examples of the commands.

Table 35-3 Sample **boot system** Commands

Boot System Command	Result
boot system flash	The first file from system flash memory is loaded.
boot system flash *filename*	IOS with the name *filename* is loaded from system flash memory.
boot system tftp *filename* **10.1.1.1**	IOS with the name *filename* is loaded from the TFTP server at address 10.1.1.1.

Finally, remember the process of upgrading the IOS? The whole point of the **boot system** commands and boot field of the configuration register is to control which IOS loads. Once a new IOS has been copied into flash memory on the router, the upgrade process has a few more steps. Add a **boot system** command to refer to the correct new file, save the configuration, and reload the router. The router will now go through the boot sequence discussed in this section, load the new IOS image, and the IOS upgrade is complete. For instance, Example 35-2 showed a router copying an IOS image into flash; that router would then also need a **boot system flash:c2900-universalk9-mz.SPA.152-4.M1.bin** command saved into the startup-config.

Verifying the IOS Image Using the show version Command

Once it is upgraded, you should verify the new IOS has loaded using the **show version** command. This command lists not only the version of software but also the source from which the router found the IOS image and the time since it loaded the IOS. As a result, the **show version** command actually identifies some key facts about the results of the previous boot process.

The **show version** command lists many other facts as well, as shown in Example 35-7. The example shows output from Router R2, which has been configured with the **boot system flash:c2900-universalk9-mz.SPA.152-4.M1.bin** command and been reloaded, migrating to use the new Version 15.2(4) IOS.

To help point out some of the many important facts in this command, the example shows many highlighted items. The following list describes each of the items in the output in the same order as they are shown in the example, top to bottom:

1. The IOS version
2. The uptime (the length of time that has passed since the last reload)
3. The reason for the last reload of IOS (reload command, power off/on, software failure)
4. The time of the last loading of IOS (if the router's clock has been set)
5. The source from which the router loaded the current IOS
6. The amount of RAM memory
7. The number and types of interfaces
8. The amount of NVRAM memory
9. The amount of flash memory
10. The configuration register's current and future setting (if different)

Key Topic

Example 35-7 show version *Command Output*

```
R2# show version
Cisco IOS Software, C2900 Software (C2900-UNIVERSALK9-M), Version 15.2(4)M1, RELEASE
  SOFTWARE (fc1)
Technical Support: http://www.cisco.com/techsupport
Copyright  1986-2012 by Cisco Systems, Inc.
Compiled Thu 26-Jul-12 20:54 by prod_rel_team

ROM: System Bootstrap, Version 15.0(1r)M15, RELEASE SOFTWARE (fc1)

R2 uptime is 44 minutes
System returned to ROM by reload at 19:44:01 UTC Tue Feb 12 2013
System restarted at 19:45:53 UTC Tue Feb 12 2013
System image file is "flash:c2900-universalk9-mz.SPA.152-4.M1.bin"
Last reload type: Normal Reload
Last reload reason: Reload Command

This product contains cryptographic features and is subject to United
States and local country laws governing import, export, transfer and

! Rest of legal disclaimer omitted
```

```
Cisco CISCO2901/K9 (revision 1.0) with 483328K/40960K bytes of memory.
Processor board ID FTX1628837T
2 Gigabit Ethernet interfaces
4 Serial(sync/async) interfaces
1 terminal line
DRAM configuration is 64 bits wide with parity enabled.
255K bytes of non-volatile configuration memory.
3425968K bytes of USB Flash usbflash1 (Read/Write)
250880K bytes of ATA System CompactFlash 0 (Read/Write)


License Info:

License UDI:

-------------------------------------------------
Device#    PID                     SN
-------------------------------------------------
*0         CISCO2901/K9            FTX1628837T


Technology Package License Information for Module:'c2900'
-----------------------------------------------------------------
Technology      Technology-package              Technology-package
                Current        Type             Next reboot
------------------------------------------------------------------
ipbase          ipbasek9       Permanent        ipbasek9
security        None           None             None
uc              None           None             None
data            None           None             None

Configuration register is 0x2102
```

Password Recovery

Suppose that you are sitting at your desk and you try to Secure Shell (SSH) or Telnet to a router. However, you cannot log in. Or, you can get into user mode but not into enable mode because you forgot the **enable secret** password. You want to recover, or at least reset the passwords, so you can get into the router and change the configuration. What can you do?

Cisco provides a way to reset the passwords on a router when sitting beside the router. With access to the router console and the ability to power the router off and back on, anyone can reset all the passwords on the router to new values.

The details differ from router model to router model. However, if you go to www.cisco.com and search for “password recovery,” within the first few hits you should see a master password

Managing Configuration Files

Cisco routers and switches happen to use two different configuration files: a startup-config file to save the configuration to use each time the device boots, and the running-config file that holds the currently used configuration for current use inside RAM. Chapter 6, "Using the Command-Line Interface," introduced those ideas, and by now, you should be used to changing the running-config file using configuration mode and saving the running-config using the **copy running-config startup-config** command.

This last of three major sections of the chapter takes the discussion of configuration files much further. It begins with a look at the traditional methods to copy configuration files outside the router or switch. It then examines more recent options to archive and restore the configuration. This section ends with a brief example of the setup process by which the router can build an initial configuration file.

Copying and Erasing Configuration Files

A good operational plan includes regular backup of the configuration files. The startup and running-config files reside in the router only, and that poses a risk. If the router configuration is never backed up to an external site and the router fails, then even after you replace the router hardware, you may have difficulty piecing a correct router configuration together based on old project notes.

The IOS **copy** command gives you a way to make a copy of the configuration, and has been around for a long time. This command lets you use any of the IFS references to network protocols, including TFTP, FTP, and SCP.

You can also just copy files to and from removable USB flash memory in the router. The USB slots on most recent models of Cisco routers allow you to insert and remove the USB flash drives with IOS running. For instance, a Cisco 2901 router has two USB flash drive slots (usbflash0: and usbflash1:). As demonstrated in Example 35-9, an engineer could easily copy the running-config file to flash.

Example 35-9 *Copying a File to USB Flash*

```
R1# copy running-config usbflash1:temp-copy-of-config
Destination filename [temp-copy-of-config]?
3159 bytes copied in 0.944 secs (3346 bytes/sec)

R1# dir usbflash1:
Directory of usbflash1:/

! lines listing other files omitted for brevity.
   74  -rw-          3159  Feb 12 2013 22:17:00 +00:00   temp-copy-of-config

7783804928 bytes total (7685111808 bytes free)
R1#
```

While useful in a lab, using USB flash to back up configuration files does not work well with thousands of devices spread around many sites. More than likely, you would back up the files to a more centralized server over the network. The next topic looks at the overall backup and restore plan for systematically backing up configurations.

Traditional Configuration Backup and Restore with the copy Command

One primary motivation of copying the configuration to an external server is to then later restore the configuration if a problem occurs. Like any backup and restore process, the configuration restore process is just as important as backing up the configuration. However, the IOS **copy** command, which has been in IOS for a long time, has an odd behavior when copying files to the running config file to restore the configuration. That odd behavior impacts how to restore the configuration rather than how to back up the configuration.

The **copy** command does not replace the running-config file when copying a configuration into RAM. Effectively, any copy into the running-config file works just as if you entered the commands in the "from" configuration file while in configuration mode. In some cases, adding the new commands does actually replace the old command; for instance, the **ip address** interface subcommand would simply replace the old value. However, with other commands, the command would not replace the old configuration but add to it instead (for instance, with IP **access-list** commands), creating a different configuration.

To drive the point home with a few examples, Figure 35-5 shows the cases that result in a replacement of the configuration versus an addition to the configuration. The figure shows commands to copy to and from a TFTP server. Note that the two commands with an asterisk beside them are the ones that effectively add the configuration.

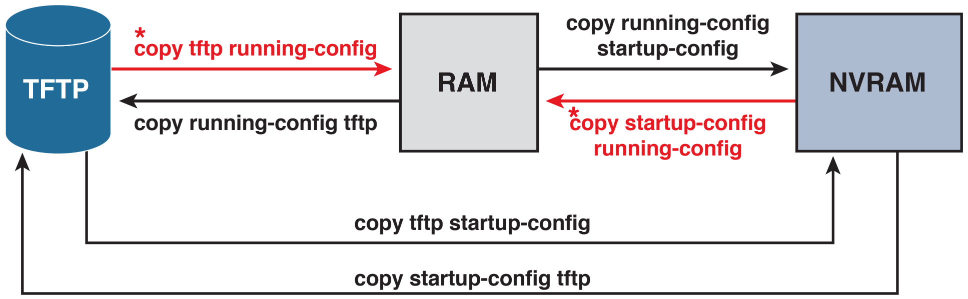

TFTP
* copy tftp running-config
copy running-config tftp
RAM
copy running-config startup-config
NVRAM
* copy startup-config running-config
copy tftp startup-config
copy startup-config tftp

Figure 35-5 *Copy into RAM (running-config) Adds Configuration Instead of Replacing*

Because of the effect of copying configurations into the running-config file, the restore process basically avoids using the **copy** command to copy a backup configuration file into running-config. The complete process, using the **copy** command, to both back up and restore configurations, works like this:

Key Topic

Step 1. To back up: Copy the running-config file to some external server; for instance, **copy running-config tftp**.

Step 2. To restore:

A. Copy the backup configuration into the startup-configuration file using the **copy** command, which replaces the startup-config file; for instance, **copy tftp startup-config**.

B. Issue the **reload** command, which reloads, or reboots, the router. That process erases all running config in RAM and then copies the startup-config into RAM as part of the reload process.

Alternatives for Configuration Backup and Restore

Cisco has improved the backup and restore process over the years beyond the basic capabilities of the IOS **copy** command. Two improvements stand out as compared to the use of the **copy** command:

Key Topic

- Create backup configurations, called *archives*, based on the use of the **archive** EXEC command. Archives can be created by command, based on a configured timer, or automatically created each time someone saves the configuration.
- Perform a restore of the archived configuration to the running-config file without requiring a reload by using the **configure replace** command.

The archival process revolves around an IOS file system called the archive. The router just needs to know where to store these configuration files. The router also needs to know whether or not to save the configuration archives automatically. Those rules define the archive—when to automatically save the configuration and where to save them. Example 35-10 shows a sample archive configuration, in which the router defines an FTP server at address 192.168.1.170 as the place to store the configurations, with username wendell and password odom. It also defines automatic backup every 1,440 minutes (that is, daily) and stores a copy of the configuration every time the configuration is saved (per the **write-memory** subcommand).

Example 35-10 *Creating a Configuration Archive*

```
R1# configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
R1(config)# archive
R1(config-archive)# path ftp://wendell:odom@192.168.1.170/
R1(config-archive)# time-period 1440
R1(config-archive)# write-memory
R1(config-archive)# ^Z
R1#
```

NOTE IOS originally used the **write memory** EXEC command to save the configuration; that command was replaced by the **copy running-config startup-config** command. The **archive** feature's **write-memory** command appears to refer to this old EXEC command.

The configuration in the example makes a great improvement over using the **copy** command. First, it improves backups by backing up the configuration automatically. It also improves the restore process because of the **configure replace** command. Basically, the **configure replace** command allows you to copy a configuration archive into the running-config file, so it completely replaces the running-config without requiring a reload of the router. Basically, the router analyzes all the configuration, does a series of comparisons, and determines what sequence of configuration commands would be required to change the configuration correctly—all without reloading the router.

To show the process, Example 35-11 shows a sequence in which a router does not have an ACL (141) at the time the archive is made. Then the user changes the configuration to add an ACL 141. Next, the **configure restore** command is used to restore the earlier archived configuration (which doesn't have ACL 141). Because the restore should replace the running-config file, the running-config should no longer have ACL 141 at the end of the process. The example also shows the hostname being changed as a more obvious confirmation that the **configure replace** command changed the configuration.

Key Topic

Example 35-11 *Replacing the Running-config with the* **configure replace** *Command*

```
R1# archive config
Writing -Oct-24-09-46-43.165-2
R1# show archive
The maximum archive configurations allowed is 10.
The next archive file will be named ftp://wendell:odom@192.168.1.170/-<timestamp>-3
 Archive #  Name
   1        ftp://wendell:odom@192.168.1.170/-Oct-24-09-21-38.865-0
   2        ftp://wendell:odom@192.168.1.170/-Oct-24-09-22-22.561-1
   3        ftp://wendell:odom@192.168.1.170/-Oct-24-09-46-43.165-2 <- Most Recent

R1# configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
R1(config)# hostname ridiculousname
ridiculousname(config)# access-list 141 permit ip host 2.2.2.2 host 3.3.3.3
ridiculousname(config)# ^Z
ridiculousname#
*Oct 24 09:47:57.189: %SYS-5-CONFIG_I: Configured from console by console

ridiculousname# configure replace ftp://wendell:odom@192.168.1.170/
  -Oct-24-09-46-43.165-2
This will apply all necessary additions and deletions
to replace the current running configuration with the
contents of the specified configuration file, which is
assumed to be a complete configuration, not a partial
configuration. Enter Y if you are sure you want to proceed. ? [no]: y
Loading -Oct-24-09-46-43.165-2 !
[OK - 6498/4096 bytes]

Loading -Oct-24-09-46-43.165-2 !
Total number of passes: 1
Rollback Done

R1# show access-list 141
R1#
```

Note that by the end of the example, the hostname has reverted back to the original name (R1) and ACL 141 is no longer configured, as expected.

Erasing Configuration Files

IOS supports three different commands to erase the startup-config file in NVRAM. The **write erase** and **erase startup-config** commands are older, whereas the **erase nvram:** command is the more recent, and recommended, command.

Note that Cisco IOS does not have a command that erases the contents of the running-config file. To clear out the running-config file, simply erase the startup-config file; then **reload** the router so that the router loads an empty startup-config file into the running-config.

Initial Configuration (Setup Mode)

Cisco IOS software supports two primary methods of giving a router or switch an initial basic configuration: configuration mode and setup mode. Setup mode leads a switch administrator through a basic configuration by using questions that prompt the administrator. Because configuration mode is required for most configuration tasks, most networking personnel quickly get comfortable with configuration mode and do not use setup at all. However, new users sometimes like to use setup mode, particularly until they become more familiar with the CLI configuration mode.

Just so you know how to get to setup mode, an engineer can get into setup mode in two ways. Figure 35-6 shows one of the methods that occurs during the boot process: If the router boots, with no initial configuration, the router asks if the user wants to enter the “initial configuration dialogue,” also known simply as setup mode. You can also enter setup mode by using the **setup** command from privileged mode.

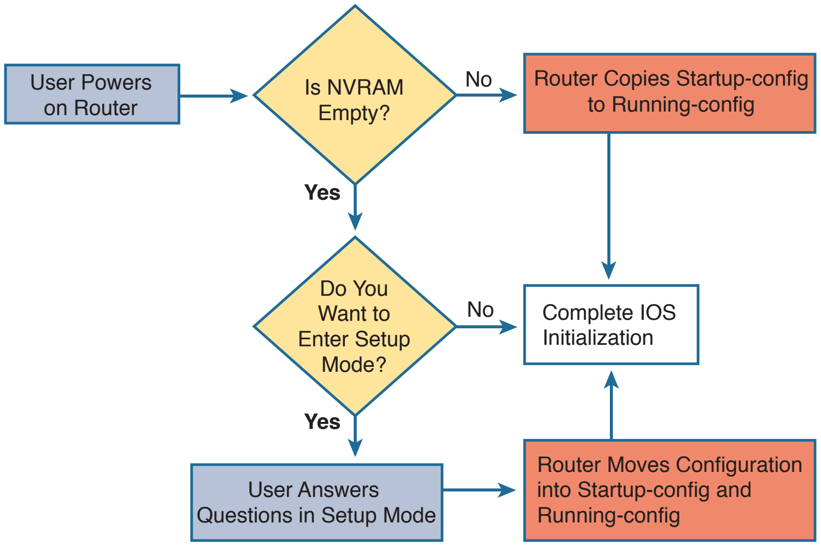

User Powers on Router
Is NVRAM Empty?
No
Router Copies Startup-config to Running-config
Yes
Do You Want to Enter Setup Mode?
No
Complete IOS Initialization
Yes
User Answers Questions in Setup Mode
Router Moves Configuration into Startup-config and Running-config

Figure 35-6 *Logic and Decisions for Entering Setup Mode After Reload*

NOTE Example 35-8, earlier in this chapter, showed the password recovery process. That process caused a router to boot while ignoring the initial configuration, causing the router to ask the user the question shown in Figure 35-6.