APPENDIX B

SWITCH Exam Updates: Version 1.1

Note: Coverage of this topic should be integrated into your study of Chapter 11, "Multilayer Switching," right before the section "Multilayer Switching with CEF."

Controlling the Automatic State of an SVI

Because a Layer 3 SVI is bound to a Layer 2 VLAN on a switch, it normally follows the state of the VLAN on that switch automatically. If the switch has at least one Layer 2 interface that is up and active on the VLAN, the Layer 3 SVI will be brought up, too. If all the Layer 2 interfaces assigned to the VLAN are down, the Layer 3 interface will be brought down.

This is the default "autostate" behavior. The idea is to bring the Layer 3 interface down so that routing protocols will cease advertising a route to the IP subnet if no active switch interfaces exist on the VLAN where the subnet exists.

When the SVI autostate feature is enabled, a Layer 3 SVI can come up only if the following three conditions are met:

- The VLAN bound to the SVI exists and is active in the VLAN database on the switch
- The SVI is not administratively shut down
- At least one Layer 2 interface is assigned to the SVI's VLAN and is in the up state, with STP forwarding

As an example, a switch has VLAN 2 defined and assigned to a variety of Layer 2 interfaces, but none of the interfaces are up. A Layer 3 SVI called interface vlan2 is then defined. Watch what happens to interface vlan2 in the following console output:

```
Switch(config)#interface vlan2
Switch(config-if)#
*Apr 21 10:13:10.949: %LINK-3-UPDOWN: Interface Vlan2, changed state to up
Switch(config-if)#
Switch(config-if)#ip address 192.168.1.1 255.255.255.0
Switch(config-if)#^Z
Switch#
```

```
Switch# show ip interface brief
Interface              IP-Address      OK? Method  Status                  Protocol
Vlan1                  unassigned      YES manual  administratively down   down
Vlan2                  192.168.1.1     YES manual  up                      down
FastEthernet1/0/1      unassigned      YES unset   down                    down
FastEthernet1/0/2      unassigned      YES unset   down                    down
```

Even before an IP address can be configured on the new SVI, the switch brings its status up but its line protocol stays down. In other words, the SVI now exists and is bound to VLAN 2, but it is unusable until at least one Layer 2 interface becomes active on VLAN 2.

In the following output, notice what happens as a PC is connected to interface FastEthernet1/0/1, which is assigned to VLAN 2:

```
Switch#
*Apr 21 10:21:31.925: %LINK-3-UPDOWN: Interface FastEthernet1/0/1, changed
  state to up
*Apr 21 10:21:32.009: %LINEPROTO-5-UPDOWN: Line protocol on Interface Vlan2,
  changed state to up
*Apr 21 10:21:32.932: %LINEPROTO-5-UPDOWN: Line protocol on Interface
  FastEthernet1/0/1, changed
state to up
Switch#
```

When the Layer 2 interface comes up, so does the line protocol of the SVI. When the PC is disconnected or powered down, the SVI is automatically taken down, as shown in the following output:

```
Switch#
*Apr 21 10:21:45.624: %LINEPROTO-5-UPDOWN: Line protocol on Interface
  FastEthernet1/0/1, changed state to
down
*Apr 21 10:21:45.624: %LINEPROTO-5-UPDOWN: Line protocol on Interface Vlan2,
  changed state to down
*Apr 21 10:21:46.622: %LINK-3-UPDOWN: Interface FastEthernet1/0/1, changed
  state to down
Switch#
```

You can override the default behavior by disabling autostate on a per-interface basis with the following command:

```
Switch(config-if)# switchport autostate exclude
```

When an interface is excluded, any influence that it might have had over the SVI state is removed. This command isn't normally used unless the interface is a special case, such as an interface where a network analyzer is connected. The analyzer would capture traffic without being an active participant in the VLAN that is assigned to the interface.

Note: Coverage of these topics should be integrated into your study of Chapter 13, "Layer 3 High Availability."

Monitoring the Network to Maximize Availability

You can (and should) design and implement a highly available network by leveraging many of the features presented in this book. For example, you can bring up redundant links between switches, redundant gateways between switches, and redundant route processors to provide as much "uptime" as possible for the end users and mission-critical applications. Once configured, these features are all automatic, requiring no further intervention.

But how will you know when some event occurs to trigger a high availability feature? Suppose a redundant link goes down for some reason. If you don't know about it and aren't able to fix it right away, then you no longer have redundant links to mitigate another future failure.

This section covers the following three important ways you can monitor the network to detect failures and gather information about switch activity:

- Syslog messages
- SNMP
- IP SLA

Syslog Messages

Catalyst switches can be configured to generate an audit trail of messages describing important events that have occurred. These system message logs (syslog) can then be collected and analyzed to determine what has happened, when it happened, and how severe the event was.

When system messages are generated, they always appear in a consistent format, as shown in Figure B-1. Each message contains the following fields:

- **Timestamp—**The date and time from the internal switch clock. By default, the amount of time that the switch has been up is used.
- **Facility Code—**A system identifier that categorizes the switch function or module that has generated the message; the facility code always begins with a percent sign.
- **Severity—**A number from 0 to 7 that indicates how important or severe the event is; a lower severity means the event is more critical.
- **Mnemonic—**A short text string that categorizes the event within the facility code.
- **Message Text—**A description of the event or condition that triggered the system message.

In Figure B-1, an event in the "System" or SYS facility has triggered the system message. The event is considered to be severity level 5. From the mnemonic CONFIG_I, you can infer that something happened with the switch configuration. Indeed, the text description says that the switch was configured by someone connected to the switch console port.

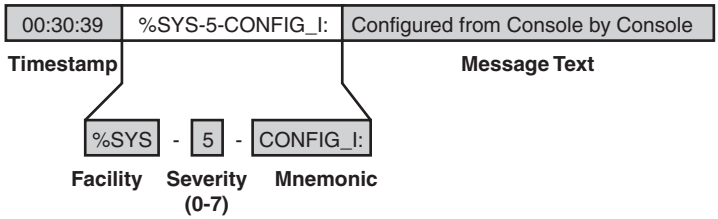

Figure B-1 *Catalyst Switch Syslog Message Format*

Generally, you should configure a switch to generate syslog messages that are occurring at or above a certain level of importance; otherwise, you might collect too much information from a switch that logs absolutely everything or too little information from a switch that logs almost nothing.

You can use the severity level to define that threshold. Figure B-2 shows each of the logging severity levels, along with a general list of the types of messages that are generated. Think of the severity levels as concentric circles. When you configure the severity level threshold on a switch, the switch will only generate logging messages that occur at that level or at any other level that is contained within it.

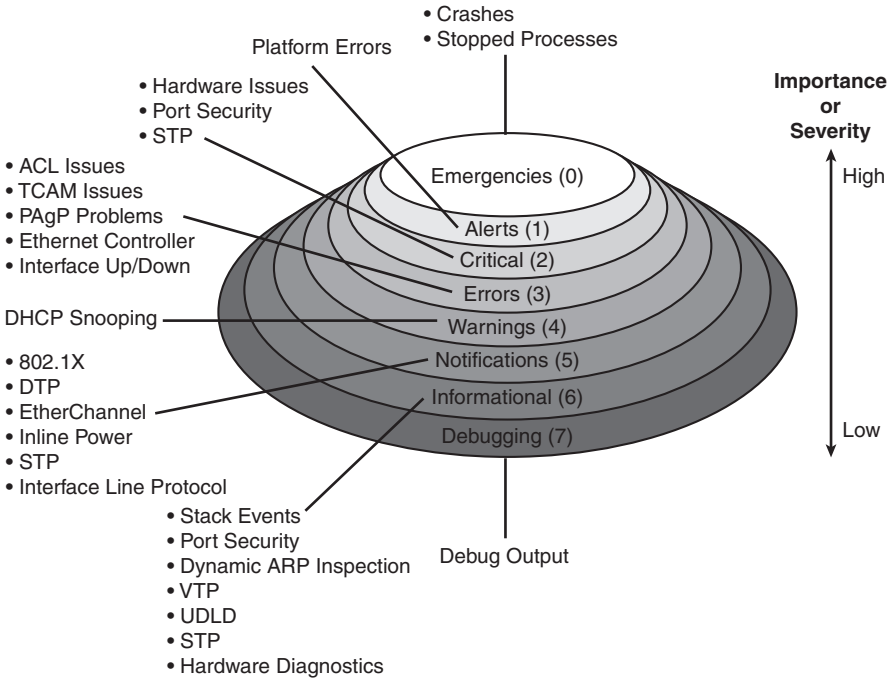

Figure B-2 *Syslog Severity Levels*

For example, if the syslog severity level is set to critical (severity level 2), the switch will generate messages in the Critical, Alerts, and Emergencies levels, but nothing else. Notice

that the severity levels are numbered such that the most urgent events are reported at level 0 and the least urgent at level 7.

Tip: You should try to have a good understanding of the severity level names and numbers, as well as their order, in case you need to identify them on the exam.

Remember that the severity numbers are opposite of what you might perceive as the actual message importance. If the exam asks about an event that has a "high" or "greater" severity, that means the severity level number will be lower.

System messages can be sent to the switch console, collected in an internal memory buffer, and sent over the network to be collected by a syslog server. The configuration commands for each of these destinations are covered in the following sections.

Logging to the Switch Console

By default, system messages are sent to the switch console port at the Debugging level. You can change the console severity level with the following command:

```
Switch(config)# logging console severity
```

The *severity* parameter can be either a severity level keyword, such as **informational**, or the corresponding numeric value (0 to 7).

Remember that syslog information can be seen on the console only when you are connected to the console port. Even then, the console isn't a very efficient way to collect and view system messages because of its low throughput. If you are connected to a switch through a Telnet or SSH session, you can redirect the console messages to your remote access session by using the **terminal monitor** command.

Logging to the Internal Buffer

Every Catalyst switch has an internal memory buffer where syslog messages can be collected. The internal buffer is an efficient way to collect messages over time. As long as the switch is powered up, the logging buffer is available.

By default, the internal logging buffer is disabled. To enable it and begin sending system messages into the buffer, you can use the following command:

```
Switch(config)# logging buffered severity
```

The *severity* parameter can be either a severity level keyword, such as **informational**, or the corresponding numeric value (0 to 7).

The logging buffer has a finite size and operates in a circular fashion. If the buffer fills, the oldest messages roll off as new ones arrive. By default, the logging buffer is 4096 bytes or characters long, which is enough space to collect 50 lines of full-length text. If you depend on the logging buffer to keep a running history of logging messages, you might need to increase its size with the following command:

```
Switch(config)# logging buffered size
```

The buffer length is set to *size* (4096 to 2147483647) bytes. Be careful not to set the length too big because the switch reserves the logging buffer space from the memory it might need for other operations.

To review the internal logging buffer at any time, you can use the **show logging** command.

Logging to a Remote Syslog Server

Syslog servers provide the most robust method of logging message collection. Messages are sent from a switch to a syslog server over the network using UDP port 514. This means that a syslog server can be located anywhere, as long as it is reachable by the switch. A syslog server can collect logs from many different devices simultaneously and can archive the logging information for a long period of time.

To identify a syslog server and begin sending logging messages to it, you can use the following commands:

```
Switch(config)# logging host
Switch(config)# logging trap severity
```

The syslog server is located at hostname or IP address *host*. You can enter the **logging** *host* command more than once if you have more than one syslog server collecting logging information. The syslog server severity level can be either a severity level keyword, such as **informational**, or the corresponding numeric value (0 to 7).

Tip: Notice that each of the logging message destinations can have a unique severity level configured. For example, you might collect messages of severity level Debugging into the internal buffer, while collecting severity level Notifications to a syslog server.

Tip: By default, a switch will generate a system message every time it detects an interface going up or down. That sounds like a good thing, until you realize that the syslog server will be receiving news of every user powering their PC on and off each day. Each link state change will generate a message at the Errors (3) severity level, as well as a line protocol state change at Notifications (5) severity level. To prevent this from happening with Access layer interfaces where end users are connected, you can use the following interface configuration command:

```
Switch(config-if)# no logging event link-status
```

Adding Timestamps to Syslog Messages

If you are watching system messages appear in real time, it's obvious what time those events have occurred. However, if you need to review messages that have been collected and archived in the internal logging buffer or on a syslog server, message timestamps become really important.

By default, Catalyst switches add a simple "uptime" timestamp to logging messages. This is a cumulative counter that shows the hours, minutes, and seconds since the switch has been booted up. Suppose you find an important event in the logs and you want to know

exactly when it occurred. With the uptime timestamp, you would have to backtrack and compute the time of the event based on how long the switch has been operating.

Even worse, as time goes on, the uptime timestamp becomes more coarse and difficult to interpret. In the following output, an interface went down 20 weeks and 2 days after the switch was booted. Someone made a configuration change 21 weeks and 3 days after the switch booted. At exactly what date and time did that occur? Who knows!

```
20w2d: %LINK-3-UPDOWN: Interface FastEthernet1/0/27, changed state to down
21w3d: %SYS-5-CONFIG_I: Configured from console by vty0 (172.25.15.246)
```

Instead, you can configure the switch to add accurate clock-like timestamps that are easily interpreted. Sometimes you also will need to correlate events in the logs of several network devices. In that case, it's important to synchronize the clocks (and timestamps) across all those devices.

Don't assume that a switch already has its internal clock set to the correct date and time. You can use the **show clock** command to find out, as in the following example:

```
Switch# show clock
*00:54:09.691 UTC Mon Mar 1 1993
Switch#
```

Here the clock has been set to its default value, and it's March 1, 1993! Clearly, that isn't useful at all.

You can use the following commands as a guideline to define the time zone and summer (daylight savings) time, and set the clock:

```
Switch(config)# clock timezone name offset-hours [offset-minutes]
Switch(config)# clock summer-time name date start-month date year hh:mm
  end-month day year hh:mm
[offset-minutes]
-OR-
Switch(config)# clock summer-time name recurring [start-week day month
  hh:mm end-week day month hh:mm [offset-minutes]
Switch(config)# exit
Switch# clock set hh:mm:ss
```

In the following example, the switch is configured for the Eastern time zone in the U.S. and the clock is set for 3:23pm. If no other parameters are given with the **clock summer-time recurring** command, U.S. daylight savings time is assumed.

```
Switch(config)# clock timezone EST -5
Switch(config)# clock summer-time EDT recurring
Switch(config)# exit
Switch# clock set 15:23:00
```

To synchronize the clocks across multiple switches in your network from common, trusted time sources, you should use the Network Time Protocol (NTP). As a simple ex-

ample, the following commands are used to enable NTP and use the NTP server at 192.43.244.18 (time.nist.gov) as an authoritative source:

```
Switch(config)# ntp server 192.43.244.18
```

After NTP is configured and enabled, you can use the **show ntp status** command to verify that the switch clock is synchronized to the NTP server.

Finally, you can use the following command to begin using the switch clock as an accurate timestamp for syslog messages:

```
Switch(config)# service timestamps log datetime [localtime] [show-timezone]
  [msec] [year]
```

Use the **localtime** keyword to use the local time zone configured on the switch; otherwise, UTC is assumed. Add the **show-timezone** keyword if you want the time zone name added to the timestamps. Use the **msec** keyword to add milliseconds and the **year** keyword to add the year to the timestamps.

In the following example, the local time zone and milliseconds have been added into the timestamps of the logging messages shown:

```
Switch(config)# service timestamps log datetime localtime show-timezone msec
Switch(config)# exit
Switch# show logging
*May  2 02:39:23.871 EDT: %DIAG-SP-6-DIAG_OK: Module 1: Passed Online Diagnostics
*May  2 02:39:27.827 EDT: %HSRP-5-STATECHANGE: Vlan62 Grp 1 state Standby -> Active
*May  2 02:41:40.431 EDT: %OIR-SP-6-INSCARD: Card inserted in slot 9, interfaces
  are now online
*May  3 08:24:13.944 EDT: %IP-4-DUPADDR: Duplicate address 10.1.2.1 on Vlan5,
  sourced by 0025.64eb.216f
*May 13 09:55:57.139 EDT: %SYS-5-CONFIG_I: Configured from console by herring on
  vty0 (10.1.1.7)
```

SNMP

The Simple Network Management Protocol (SNMP) enables a network device to share information about itself and its activities. A complete SNMP system consists of the following parts:

- **SNMP Manager—**A network management system that uses SNMP to poll and receive data from any number of network devices. The SNMP Manager usually is an application that runs in a central location.
- **SNMP Agent—**A process that runs on the network device being monitored. All types of data are gathered by the device itself and stored in a local database. The agent can then respond to SNMP polls and queries with information from the database, and it can send unsolicited alerts or "traps" to an SNMP Manager.

In the case of Catalyst switches in the network, each switch automatically collects data about itself, its resources, and each of its interfaces. This data is stored in a Management Information Base (MIB) database in memory and is updated in real time.

The MIB is organized in a structured, hierarchical fashion, forming a tree structure. In fact, the entire MIB is really a collection of variables that are stored in individual, more granular MIBs that form the branches of the tree. Each MIB is based on the Abstract Syntax Notation 1 (ASN.1) language. Each variable in the MIB is referenced by an object identifier (OID), which is a long string of concatenated indexes that follow the path from the root of the tree all the way to the variable's location.

Fortunately, only the SNMP manager and agent need to be concerned with interpreting the MIBs. As far as the SWITCH exam and course go, you should just be aware that the MIB structure exists and that it contains everything about a switch that can be monitored.

To see any of the MIB data, an SNMP manager must send an SNMP poll or query to the switch. The query contains the OID of the specific variable being requested so that the agent running on the switch knows what information to return. An SNMP manager can use the following mechanisms to communicate with an SNMP agent, all over UDP port 161:

- **Get Request—**The value of one specific MIB variable is needed.
- **Get Next Request—**The next or subsequent value following an initial Get Request is needed.
- **Get Bulk Request—**Whole tables or lists of values in a MIB variable are needed.
- **Set Request—**A specific MIB variable needs to be set to a value.

SNMP polls or requests are usually sent by the SNMP manager at periodic intervals. This makes real-time monitoring difficult because changing variables won't be noticed until the next poll cycle. However, SNMP agents can send unsolicited alerts to notify the SNMP manager of real-time events at any time. Alerts can be sent using the following mechanisms over UDP port 162:

- **SNMP Trap—**News of an event (interface state change, device failure, and so on) is sent without any acknowledgement that the trap has been received.
- **Inform Request—**News of an event is sent to an SNMP manager, and the manager is required to acknowledge receipt by echoing the request back to the agent.

As network management has evolved, SNMP has developed into three distinct versions. The original, SNMP version 1 (SNMPv1), is defined in RFC1157. It uses simple one-variable Get and Set requests, along with simple SNMP traps. SNMP managers can gain access to SNMP agents by matching a simple "community" text string. When a manager wants to read or write a MIB variable on a device, it sends the community string in the clear, as part of the request. The request is granted if that community string matches the agent's community string.

In theory, only managers and agents belonging to the same community should be able to communicate. In practice, any device has the potential to read or write variables to an agent's MIB database by sending the right community string, whether it's a legitimate SNMP manager or not. This creates a huge security hole in SNMPv1.

The second version of SNMP, SNMPv2C (RFC 1901), was developed to address some efficiency and security concerns. For example, SNMPv2c adds 64-bit variable counters, extending the useful range of values over the 32-bit counters used in SNMPv1. In addition, SNMPv2C offers the Bulk Request, making MIB data retrieval more efficient. It also offers Inform Requests, which make real-time alerts more reliable by requiring confirmation of receipt.

Despite the intentions of its developers, SNMPv2C does not address any security concerns over that of SNMPv1. SNMPv2C does offer 64-bit variable counters, which extend the capability to keep track of very large numbers, such as byte counters found on high-speed interfaces. With SNMPv2C, MIB variables can be obtained in a bulk form with a single request. In addition, event notifications sent from an SNMPv2C agent can be in the form of SNMP traps or inform requests. The latter form requires an acknowledgement from the SNMP manager that the inform message was received.

The third generation of SNMP, SNMPv3, is defined in RFCs 3410 through 3415. It addresses the security features that are lacking in the earlier versions. SNMPv3 can authenticate SNMP managers through usernames. When usernames are configured on the SNMP agent of a switch, they can be organized into SNMPv3 group names.

Each SNMPv3 group is defined with a security level that describes the extent to which the SNMP data will be protected. Data packets can be authenticated to preserve their integrity, encrypted to obscure their contents, or both. The following security levels are available. The naming scheme uses “auth” to represent packet authentication and “priv” to represent data privacy or encryption.

- **noAuthNoPriv—**SNMP packets are neither authenticated nor encrypted.
- **authNoPriv—**SNMP packets are authenticated but not encrypted.
- **authPriv—**SNMP packets are authenticated and encrypted.

As a best practice, you should use SNMPv3 to leverage its superior security features whenever possible. If you must use SNMPv1 for a device, you should configure the switch to limit SNMP access to a read-only role. Never permit read-write access because the simple community string authentication can be exploited to make unexpected changes to a switch configuration.

Catalyst switches offer one additional means of limiting SNMP access—an access list can be configured to permit only specific SNMP manager IP addresses. You should configure and apply an access list to your SNMP configurations whenever possible.

Because SNMP is a universal method for monitoring all sorts of network devices, it isn’t unique to LAN switches. Therefore, you should understand the basics of how SNMP works, the differences between the different SNMP versions, and how you might apply SNMP to monitor a switched network. You can use Table B-1 as a memory aid for your exam study.

Table B-1 *Comparison of SNMP Versions and Their Features*

Version	Authentication	Data Protection	Unique Features
SNMPv1	Community string	None	32-bit counters
SNMPv2c	Community string	None	Added Bulk Request and Inform Request message types, 64-bit counters
SNMPv3	Username authentication	Hash-based MAC (SHA or MD5) DES, 3DES, AES (128, 192, 256-bit) encryption	Added user authentication, data integrity, and encryption

Configuring SNMPv1 and SNMPv2C

You should be familiar with the basic SNMP configuration. Fortunately, this involves just a few commands, as follows:

```
Switch(config)# access-list access-list-number permit ip-addr
Switch(config)# snmp-server community string [ro | rw] [access-list-number]
!
Switch(config)# snmp-server host host-address community-string [trap-type]
```

First, define a standard IP access list that permits only the IP addresses of your SNMP agent machines. Then apply that access list to the SNMPv1 community string with the **snmp-server community** command. Use the **ro** keyword to allow read-only access by the SNMP manager; otherwise, use the **rw** keyword to allow both read and write access.

Finally, use the **snmp-server host** command to identify the IP address of the SNMP manager where SNMP traps will be sent. By default, all types of traps are sent. You can use the **?** key in place of *trap-type* to see a list of the available trap types.

In Example B-1, the switch is configured to allow SNMP polling from network management stations at 192.168.3.99 and 192.168.100.4. The community string **MonitorIt** is used to authenticate the SNMP requests. All possible SNMP traps are sent to 192.168.3.99.

Example B-1 *Configuring SNMPv1 Access*

```
Switch(config)# access-list 10 permit 192.168.3.99
Switch(config)# access-list 10 permit 192.168.100.4
Switch(config)# snmp-server community MonitorIt ro 10
Switch(config)# snmp-server host 192.168.3.99 MonitorIt
```

Configuring SNMPv2C

Configuring SNMPv2C is similar to configuring SNMPv1. The only difference is with SNMP trap or inform configuration. You can use the following commands to configure basic SNMPv2C operation:

```
Switch(config)# access-list access-list-number permit ip-addr
Switch(config)# snmp-server community string [ro | rw] [access-list-number]
!
Switch(config)# snmp-server host host-address [informs] version 2c community-string
```

In the **snmp-server host** command, use the **version 2c** keywords to identify SNMPv2C operation. By default, regular SNMP traps are sent. To use inform requests instead, add the **informs** keyword.

Configuring SNMPv3

SNMPv3 configuration is a bit more involved than versions 1 or 2C, due mainly to the additional security features.

```
Switch(config)# access-list access-list-number permit ip-addr
!
Switch(config)# snmp-server group group-name v3 {noauth | auth | priv}
Switch(config)# snmp-server user user-name group-name v3 auth {md5 | sha}
  auth-password priv {des | 3des | aes {128 | 192 | 256} priv-password [access-
  list-number]
!
Switch(config)# snmp-server host host-address [informs] version 3 {noauth | auth |
  priv} user-name [trap-type]
```

First, use the **snmp-server group** command to define a *group-name* that will set the security level policies for SNMPv3 users. The security level is defined by the **noauth** (no packet authentication or encryption), **auth** (packets are authenticated but not encrypted), or **priv** (packets are both authenticated and encrypted) keyword. Only the security policy is defined in the group; no passwords or keys are required yet.

Tip: The SNMPv3 **priv** keyword and packet encryption can be used only if the switch is running a cryptographic version of its Cisco IOS software image. The **auth** keyword and packet authentication can be used regardless.

Next, define a username that an SNMP manager will use to communicate with the switch. Use the **snmp-server user** command to define the *user-name* and associate it with the SNMPv3 *group-name*. The **v3** keyword configures the user to use SNMPv3.

The SNMPv3 user must also have some specifics added to its security policy. Use the **auth** keyword to define either MD5 or SHA as the packet authentication method, along with the *auth-password* text string that will be used in the hash computation. The **priv** keyword defines the encryption method (DES, 3DES, or AES 128/192/256-bit) and the *priv-password* text string that will be used in the encryption algorithm.

The same SNMPv3 username, authentication method and password, and encryption method and password must also be defined on the SNMP manager so it can successfully talk to the switch.

Finally, you can use the **snmp-server host** command to identify the SNMP manager that will receive either traps or informs. The switch can use SNMPv3 to send traps and informs, using the security parameters that are defined for the SNMPv3 *user-name*.

In Example B-2, a switch is configured for SNMPv3 operation. Access list 10 permits only stations at 192.168.3.99 and 192.168.100.4 with SNMP access. SNMPv3 access is defined for a group named NetOps using the **priv** (authentication and encryption) security level. One SNMPv3 user named mymonitor is defined; the network management station will use that username when it polls the switch for information. The username will require SHA packet authentication and AES-128 encryption, using the s3cr3tauth and s3cr3tpr1v passwords, respectively.

Finally, SNMPv3 informs will be used to send alerts to station 192.168.3.99 using the **priv** security level and username mymonitor.

Example B-2 *Configuring SNMPv3 Access*

```
Switch(config)# access-list 10 permit 192.168.3.99
Switch(config)# access-list 10 permit 192.168.100.4
Switch(config)# snmp-server group NetOps v3 priv
Switch(config)# snmp-server user mymonitor NetOps v3 auth sha s3cr3tauth priv aes
  128 s3cr3tpr1v 10
Switch(config)# snmp-server host 192.168.3.99 informs version 3 priv mymonitor
```

IP SLA

The Cisco IOS IP Service Level Agreement (IP SLA) feature can be used to gather realistic information about how specific types of traffic are being handled end-to-end across a network. To do this, an IP SLA device runs a preconfigured test and generates traffic that is destined for a far-end device. As the far end responds with packets that are received back at the source, IP SLA gathers data about what happened along the way.

IP SLA can be configured to perform a variety of tests. The simplest test involves ICMP echo packets that are sent toward a target address, as shown in Figure B-3. If the target answers with ICMP echo replies, IP SLA can then assess how well the source and destination were able to communicate. In this case, the echo failures (packet loss) and round trip transit (RTT) times are calculated, as shown in the following example:

```
Switch# show ip sla statistics aggregated
Round Trip Time (RTT) for       Index 1
Type of operation: icmp-echo
Start Time Index: 15:10:17.665 EDT Fri May 21 2010
RTT Values
        Number Of RTT: 24
        RTT Min/Avg/Max: 1/1/4 ms
Number of successes: 24
Number of failures: 0
```

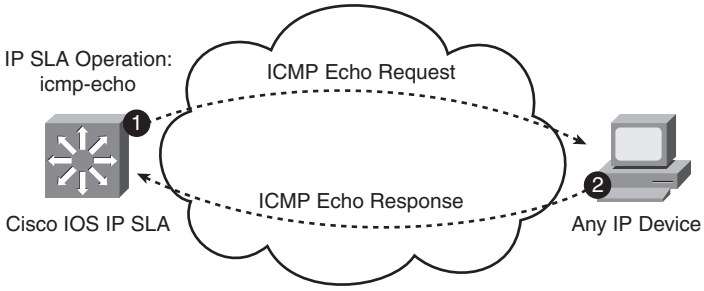

Figure B-3 *IP SLA ICMP Echo Test Operation*

For the ICMP echo test, IP SLA can use any live device at the far end—after all, most networked devices will reply when they are pinged. IP SLA also can test some network protocols, such as DNS, by sending requests to a server at the far end. Cisco IOS is needed only at the source of the IP SLA test because the far end is simply responding to ordinary request packets.

However, IP SLA is capable of running much more sophisticated tests. Table B-2 shows some sample test operations that are available with IP SLA.

Table B-2 *IP SLA Test Operations*

Test Type	Description	IP SLA Required on Target?
icmp-echo	ICMP Echo response time	No
path-echo	Hop-by-hop and end-to-end response times over path discovered from ICMP Echo	No
path-jitter	Hop-by-hop jitter over ICMP Echo path	Yes
dns	DNS query response time	No
dhcp	DHCP IP address request response time	No
ftp	FTP file retrieval response time	No
http	Web page retrieval response time	No
udp-echo	End-to-end response time of UDP echo	No
udp-jitter	Round trip delay, one-way delay, one-way jitter, one-way packet loss, and connectivity using UDP packets	Yes
tcp-connect	Response time to build a TCP connection with a host	No

To leverage its full capabilities, Cisco IOS IP SLA must be available on both the source and the target devices, as shown in Figure B-4. The source device handles the test scheduling and sets up each test over a special IP SLA control connection with the target device. The source generates the traffic involved in a test operation and analyzes the results as packets return from the target. The target end has a simpler role: respond to the incoming test packets. In fact, the target device is called an *IP SLA Responder*.

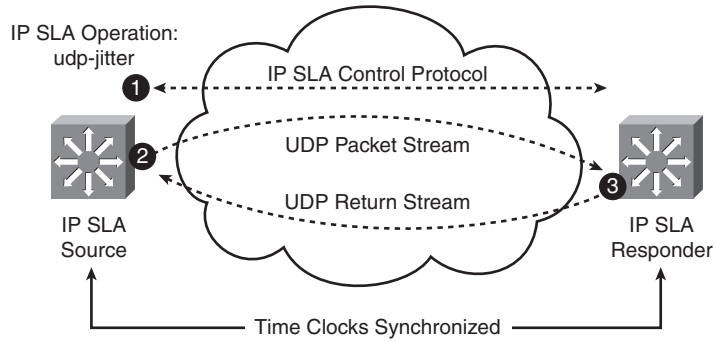

Figure B-4 *IP SLA UDP Jitter Test Operation*

The responder must also add timestamps to the packets it sends, to flag the time a test packet arrived and the time it left the responder. The idea is to account for any latency incurred while the responder is processing the test packets. For this to work accurately, both the source and responder must synchronize their clocks through NTP.

An IP SLA source device can schedule and keep track of multiple test operations. For example, an ICMP echo operation might run against target 10.1.1.1, while UDP jitter operations are running against targets 10.2.2.2, 10.3.3.3, and 10.4.4.4. Each test runs independently, at a configured frequency and duration.

Tip: To set up an IP SLA operation, the Cisco IP SLA source device begins by opening a control connection to the IP SLA responder over UDP port 1967. The source uses the control connection to inform the responder to begin listening on an additional port where the actual IP SLA test operation will take place.

What in the world does this have to do with LAN switching, and why would you want to run IP SLA on a Catalyst switch anyway? Here's a two-fold answer:

- IP SLA will likely appear on your SWITCH exam
- IP SLA is actually a useful tool in a switched campus network

To run live tests and take useful measurements without IP SLA, you would need to place some sort of probe devices at various locations in the network—all managed from a central system. With IP SLA, you don't need probes at all! Wherever you have a Catalyst switch, you already have an IP SLA "probe."

By leveraging IP SLA test operations, you can take advantage of some fancy features:

- Generate SNMP traps when certain test thresholds are exceeded
- Schedule further IP SLA tests automatically when test thresholds are crossed
- Track an IP SLA test to trigger a next-hop gateway redundancy protocol, such as HSRP
- Gather voice quality measurements from all over a network

Configuring IP SLA

You can use the following configuration steps to define and run an IP SLA test operation:

Step 1. Enable the IP SLA responder on the target switch.

```
Switch(config)# ip sla responder
```

By default, the IP SLA responder is disabled. If the IP SLA operation will involve jitter or time-critical measurements, the responder should be enabled on the target switch.

Step 2. Define a new IP SLA operation on the source switch.

```
Switch(config)# ip sla operation-number
```

The *operation-number* is an arbitrary index that can range from 1 to a very large number. This number uniquely identifies the test.

Step 3. Select the type of test operation to perform.

```
Switch(config-ip-sla)# test-type parameters...
```

The *test-type* keyword can be one of the following:

dhcp, **dns**, **ethernet**, **ftp**, **http**, **icmp-echo**, **mpls**, **path-echo**, **path-jitter**, **slm**, **tcp-connect**, **udp-echo**, or **udp-jitter**

The list of parameters following the *test-type* varies according to the test operation. As an example, consider the following **icmp-echo** operation syntax:

```
Switch(config-ip-sla)# icmp-echo destination-ip-addr [source-ip-addr]
```

The parameters are simple—a destination address to ping and an optional source address to use. If a switch has several Layer 3 interfaces, you can specify which one of their IP address to use as the source of the test packets.

As another example, the **udp-jitter** command is useful for testing time-critical traffic paths through a switched network. The command syntax is a little more complex, as follows:

```
Switch(config-ip-sla)# udp-jitter destination-ip-addr dest-udp-port
  [source-ip source-ip-addr] [source-port source-udp-port] [num-packets
  number-of-packets] [interval packet-interval]
```

In addition to the source and destination IP addresses, you can define the UDP port numbers that will be used for the packet stream. By default, 10 packets spaced at 20 milliseconds will be sent. You can override that by specifying the **num-packets** and **interval** keywords.

As an alternative, you can configure the **udp-jitter** operation to test Voice over IP (VoIP) call quality. To do this, the **udp-jitter** command must include the **codec** keyword and a codec definition. The IP SLA operation will then simulate a real-time stream of voice traffic using a specific codec. In this way, you can tailor the test to fit the type of calls that are actually being used in the network.

You can define the UDP jitter codec operation by using the following command syntax:

```
Switch(config-ip-sla)# udp-jitter destination-ip-addr dest-udp-port
  codec {g711alaw | g711ulaw | g729a}
```

There are other keywords and parameters you can add to the command, but those are beyond the scope of this book. By default, 1000 packets are sent, 20 milliseconds apart.

Step 4. Schedule the test operation.

```
Switch(config)# ip sla schedule operation-number [life {forever |
  seconds}] [start-time {hh:mm[:ss] [month day | day month] | pending |
  now | after hh:mm:ss}] [ageout seconds] [recurring]
```

In a nutshell, the command tells the switch when to start the test, how long to let it run, and how long to keep the data that is collected.

Set the lifetime with the **life** keyword: **forever** means the operation will keep running forever, until you manually remove it. Otherwise, specify how many seconds it will run. By default, an IP SLA scheduled operation will run for 3600 seconds (one hour).

Set the start time with the **start-time** keyword. You can define the start time as a specific time or date, after a delay with the **after** keyword, or right now with the **now** keyword.

By default, the test statistics are collected and held in memory indefinitely. You can use the **ageout** keyword to specify how many seconds elapse before the data is aged out.

The **recurring** keyword can be used to schedule the test operation to run at the same time each day, as long as you have defined the starting time with *hh:mm:ss*, too.

Tip: Be aware that the IP SLA operation command syntax has changed along the way. In Cisco IOS releases 12.2(33) and later, the syntax is as shown in steps 2 through 4.

Prior to 12.2(33), the commands in steps 2 through 4 included additional keywords, as follows:

Step 2. `ip sla monitor` *operation-number*

Step 3. `type` *test-type*

Step 4. `ip sla monitor schedule` *operation-number*

It isn't clear which version of the IP SLA commands are used in the SWITCH exam; just be prepared to see the syntax in either form.

Using IP SLA

After you have configured an IP SLA operation, you can verify the configuration with the **show ip sla configuration** [*operation-number*] command. As an example, the following configuration commands are used to define IP SLA operation 100—an ICMP echo test that pings target 172.25.226.1 every 5 seconds.

```
Switch(config)# ip sla 100
Switch(config-ip-sla)# icmp-echo 172.25.226.1
Switch(config-ip-sla)# frequency 5
Switch(config-ip-sla)# exit
Switch(config)# ip sla schedule 100 life forever start-time now
```

Example B-3 shows the output of the **show ip sla configuration** command.

Example B-3 *Displaying the Current IP SLA Configuration*

```
Switch# show ip sla configuration
IP SLAs, Infrastructure Engine-II
Entry number: 100
Owner:
Tag:
Type of operation to perform: echo
Target address: 172.25.226.1
Source address: 0.0.0.0
Request size (ARR data portion): 28
Operation timeout (milliseconds): 5000
Type Of Service parameters: 0x0
Verify data: No
Vrf Name:
Schedule:
    Operation frequency (seconds): 5
    Next Scheduled Start Time: Start Time already passed
```

```
    Group Scheduled : FALSE
    Randomly Scheduled : FALSE
    Life (seconds): Forever
    Entry Ageout (seconds): never
    Recurring (Starting Everyday): FALSE
    Status of entry (SNMP RowStatus): Active
Threshold (milliseconds): 5000
Distribution Statistics:
    Number of statistic hours kept: 2
    Number of statistic distribution buckets kept: 1
    Statistic distribution interval (milliseconds): 20
History Statistics:
    Number of history Lives kept: 0
    Number of history Buckets kept: 15
    History Filter Type: None
Enhanced History:
```

You can use the **show ip sla statistics** [**aggregated**] [*operation-number*] command to display the IP SLA test analysis. By default, the most recent test results are shown. You can add the **aggregated** keyword to show a summary of the data gathered over the life of the operation. Example B-4 shows the statistics gathered for ICMP echo operation 100.

Example B-4 *Displaying IP SLA Statistics*

```
Switch# show ip sla statistics 100
Round Trip Time (RTT) for       Index 100
        Latest RTT: 1 ms
Latest operation start time: 15:52:00.834 EDT Fri May 28 2010
Latest operation return code: OK
Number of successes: 117
Number of failures: 0
Operation time to live: Forever

Switch# show ip sla statistics aggregated 100
Round Trip Time (RTT) for       Index 100
Type of operation: icmp-echo
Start Time Index: 15:43:55.842 EDT Fri May 28 2010
RTT Values
        Number Of RTT: 121
        RTT Min/Avg/Max: 1/1/4 ms
Number of successes: 121
Number of failures: 0
```

It isn't too difficult to configure an IP SLA operation manually and check the results every now and then. But does IP SLA have any greater use? Yes, you can also use an IP SLA op-

eration to make some other switch features change behavior automatically, without any other intervention.

For example, HSRP can track the status of an IP SLA operation to automatically decrement the priority value when the target device stops answering ICMP echo packets. To do this, begin by using the **track** command to define a unique track *object-number* index that will be bound to the IP SLA operation number.

```
Switch(config)# track object-number ip sla operation-number {state | reachability}
```

You can use the **state** keyword to track the return code or state of the IP SLA operation; the state is up if the IP SLA test was successful or down if it wasn't. The **reachability** keyword is slightly different—the result is up if the IP SLA operation is successful or has risen above a threshold; otherwise, the reachability is down.

Next, configure the HSRP standby group to use the tracked object to control the priority decrement value. As long as the tracked object (the IP SLA operation) is up or successful, the HSRP priority stays unchanged. If the tracked object is down, then the HSRP priority is decremented by *decrement-value* (default 10).

```
Switch(config-if)# standby group track object-number decrement decrement-value
```

In Example B-5, SwitchA and SwitchB are configured as an HSRP pair, sharing gateway address 192.168.1.1. SwitchA has a higher priority than SwitchB, so it is normally the active gateway. However, it is configured to ping an upstream router at 192.168.70.1 every five seconds; if that router doesn't respond, SwitchA will decrement its HSRP priority by 50, permitting SwitchB to take over.

Example B-5 *Tracking an IP SLA Operation in an HSRP Group*

```
Switch(config)# ip sla 10
Switch(config-ip-sla)# icmp-echo 192.168.70.1
Switch(config-ip-sla)# frequency 5
Switch(config-ip-sla)# exit
Switch(config)# ip sla schedule 10 life forever start-time now

Switch(config)# track 1 ip sla 10 reachability

Switch(config)# interface vlan10
Switch(config-if)# ip address 192.168.1.3 255.255.255.0
Switch(config-if)# standby 1 priority 200
Switch(config-if)# standby 1 track 1 decrement 50
Switch(config-if)# no shutdown
```

In some cases, you might need many IP SLA operations to take many measurements in a network. For example, you could use UDP jitter operations to measure voice call quality across many different parts of the network. Manually configuring and monitoring more than a few IP SLA operations can become overwhelming and impractical. Instead, you can leverage a network management application that can set up and monitor IP SLA tests automatically. To do this, the network management system needs SNMP read and write access to each switch that will use IP SLA. Tests are configured by writing to the IP SLA MIB, and results are gathered by reading the MIB.