

Debug Commands

This chapter contains an alphabetical listing of the **debug** commands and their descriptions. Documentation for each command includes a brief description of its use, command syntax, usage guidelines, sample output, and a description of that output.

Output formats vary with each **debug** command. Some commands generate a single line of output per packet, whereas others generate multiple lines of output per packet. Some generate large amounts of output; others generate only occasional output. Some generate lines of text, and others generate information in field format. Thus, the way **debug** command output is documented also varies. For example, the output for **debug** commands that generate lines of text is usually described line by line, and the output for **debug** commands that generate information in field format is usually described in tables.

By default, the network server sends the output from the **debug** commands to the console terminal. Sending output to a terminal (virtual console) produces less overhead than sending it to the console. Use the privileged EXEC command **terminal monitor** to send output to a terminal. For more information about redirecting output, see the “Using Debug Commands” chapter.

debug aaa accounting

Use the **debug aaa accounting** EXEC command to display information on accountable events as they occur. Use the **no** form of the command to disable debugging output.

[no] debug aaa accounting

Usage Guidelines

The information displayed by the **debug aaa accounting** command is independent of the accounting protocol used to transfer the accounting information to a server. Use the **debug tacacs** and **debug radius** protocol specific commands to get more detailed information about protocol-level issues.

You can also use the **show accounting** command to step through all active sessions and to print all the accounting records for actively accounted functions. The **show accounting** command allows you to display the active “accountable events” on the system. It provides systems administrators a quick look at what is going on, and may also be useful for collecting information in the event of a data loss of some kind on the accounting server. The **show accounting** command displays additional data on the internal state of the Authentication, Authorization, and Accounting (AAA) security system if **debug aaa accounting** is turned on as well.

Sample Display

Figure 2-1 shows sample output from the **debug aaa accounting** command.

Figure 2-1 Sample Debug AAA Accounting Output

```
router# debug aaa accounting
AAA Accounting debugging is on

16:49:21: AAA/ACCT: EXEC acct start, line 10
16:49:32: AAA/ACCT: Connect start, line 10, glare
16:49:47: AAA/ACCT: Connection acct stop:
task_id=70 service=exec port=10 protocol=telnet address=172.31.3.78 cmd=glare
bytes_in=308 bytes_out=76 paks_in=45 paks_out=54 elapsed_time=14
```

Related Commands

debug aaa authentication

debug aaa authorization

debug radius

debug tacacs

debug aaa authentication

Use the **debug aaa authentication** EXEC command to display information on AAA/Terminal Access Controller Access Control System Plus (TACACS+) authentication. Use the **no** form of the command to disable debugging output.

[no] debug aaa authentication

Usage Guidelines

Use this command to see what methods of authentication are being used and what the results of these methods are.

Sample Display

Figure 2-2 shows sample **debug aaa authentication** output. A single EXEC login that uses the “default” method list and the first method, TACACS+, is displayed. The TACACS+ server sends a GETUSER request to prompt for the username and then a GETPASS request to prompt for the password, and finally a PASS response to indicate a successful login. The number 50996740 is the session ID, which is unique for each authentication. Use this ID number to distinguish between different authentications if several are occurring concurrently.

Figure 2-2 Sample Debug AAA Authentication Output

```
router# debug aaa authentication

6:50:12: AAA/AUTHEN: create_user user='' ruser='' port='tty19' rem_addr='172.31.60.15'
authen_type=1 service=1 priv=1
6:50:12: AAA/AUTHEN/START (0): port='tty19' list='' action=LOGIN service=LOGIN
6:50:12: AAA/AUTHEN/START (0): using "default" list
6:50:12: AAA/AUTHEN/START (50996740): Method=TACACS+
6:50:12: TAC+ (50996740): received authen response status = GETUSER
6:50:12: AAA/AUTHEN (50996740): status = GETUSER
6:50:15: AAA/AUTHEN/CONT (50996740): continue_login
6:50:15: AAA/AUTHEN (50996740): status = GETUSER
6:50:15: AAA/AUTHEN (50996740): Method=TACACS+
6:50:15: TAC+: send AUTHEN/CONT packet
6:50:15: TAC+ (50996740): received authen response status = GETPASS
6:50:15: AAA/AUTHEN (50996740): status = GETPASS
6:50:20: AAA/AUTHEN/CONT (50996740): continue_login
6:50:20: AAA/AUTHEN (50996740): status = GETPASS
6:50:20: AAA/AUTHEN (50996740): Method=TACACS+
6:50:20: TAC+: send AUTHEN/CONT packet
6:50:20: TAC+ (50996740): received authen response status = PASS
6:50:20: AAA/AUTHEN (50996740): status = PASS
```

debug aaa authorization

Use the **debug aaa authorization** EXEC command to display information on AAA/TACACS+ authorization. Use the **no** form of the command to disable debugging output.

[no] debug aaa authorization

Usage Guidelines

Use this command to see what methods of authorization are being used and what the results of these methods are.

Sample Display

Figure 2-3 shows sample **debug aaa authorization** output. In this display, an EXEC authorization for user “carrel” is performed. On the first line, the username is authorized. On the second and third lines, the AV (attribute value) pairs are authorized. The debug output displays a line for each attribute value pair that is authenticated. Next, the display indicates the authorization method used. The final line in the display indicates the status of the authorization process, in this case, a failure.

Figure 2-3 Sample Debug AAA Authorization Output

```
2:23:21: AAA/AUTHOR (0): user='carrel'  
2:23:21: AAA/AUTHOR (0): send AV service=shell  
2:23:21: AAA/AUTHOR (0): send AV cmd*  
2:23:21: AAA/AUTHOR (342885561): Method=TACACS+  
2:23:21: AAA/AUTHOR/TAC+ (342885561): user=carrel  
2:23:21: AAA/AUTHOR/TAC+ (342885561): send AV service=shell  
2:23:21: AAA/AUTHOR/TAC+ (342885561): send AV cmd*  
2:23:21: AAA/AUTHOR (342885561): Post authorization status = FAIL
```

The **aaa authorization** command causes a request packet containing a series of attribute value pairs to be sent to the TACACS daemon as part of the authorization process. The daemon responds in one of three ways:

- Accepts the request as is
- Makes changes to the request
- Refuses the request, thereby refusing authorization

Table 2-1 describes attribute value pairs associated with the **aaa authorization** command that may show up in the debug output.

Note Registered users can find more information about TACACS+ and attribute pairs on Cisco Information Online (CIO). Access to CIO is available through the World Wide Web at <http://www.cisco.com/> or through a telnet connection to cio.cisco.com.

Table 2-1 Attribute Value Pairs for Authorization

Attribute Value	Description
service=arap	Authorization for AppleTalk Remote Access is being requested.
service=shell	Authorization for EXEC startup and command authorization is being requested.
service=ppp	Authorization for PPP is being requested.
service=slip	Authorization for SLIP is being requested.
protocol=lcp	Authorization for LCP is being requested (lower layer of PPP).
protocol=ip	Used with service=slip and service=slip to indicate which protocol layer is being authorized.
protocol=ipx	Used with service=ppp to indicate which protocol layer is being authorized.
protocol=atalk	Used with service=ppp or service=arap to indicate which protocol layer is being authorized.
protocol=vines	Used with service=ppp for VINES over PPP.
protocol=unknown	Used for undefined or unsupported conditions.
cmd=x	Used with service=shell, if cmd=NULL, this is an authorization request to start an EXEC. If cmd is not NULL, this is a command authorization request and will contain the name of the command being authorized. For example, cmd=telnet.
cmd-arg=x	Used with service=shell. When performing command authorization, the name of the command is given by a cmd=x pair for each argument listed. For example, cmd-arg=archie.sura.net.
acl=x	Used with service=shell and service=arap. For ARA, this pair contains an access list number. For service=shell, this pair contains an access class number. For example, acl=2.
inacl=x	Used with service=ppp and protocol=ip. Contains an IP input access list for SLIP or PPP/IP. For example, inacl=2.
outacl=x	Used with service=ppp and protocol=ip. Contains an IP output access list for SLIP or PPP/IP. For example, outacl=4.
addr=x	Used with service=slip, service=ppp, and protocol=ip. Contains the IP address that the remote host should use when connecting via SLIP or PPP/IP. For example, addr=172.30.23.11.
routing=x	Used with service=slip, service=ppp, and protocol=ip. Equivalent in function to the /routing flag in SLIP and PPP commands. Can either be true or false. For example, routing=true.
timeout=x	Used with service=arap. The number of minutes before an ARA session disconnects. For example, timeout=60.
autocmd=x	Used with service=shell and cmd=NULL. Specifies an autocommand to be executed at EXEC startup. For example, autocmd=telnet foo.com.
noescape=x	Used with service=shell and cmd=NULL. Specifies a noescape option to the username configuration command. Can be either true or false. For example, noescape=true.
nohangup=x	Used with service=shell and cmd=NULL. Specifies a nohangup option to the username configuration command. Can be either true or false. For example, nohangup=false.
priv-lvl=x	Used with service=shell and cmd=NULL. Specifies the current privilege level for command authorization as a number from 0 to 15. For example, priv-lvl=15.

Table 2-1 Attribute Value Pairs for Authorization (Continued)

Attribute Value	Description
zonelist= <i>x</i>	Used with service=arap. Specifies an AppleTalk zonelist for ARA. For example, zonelist=5.
addr-pool= <i>x</i>	Used with service=ppp and protocol=ip. Specifies the name of a local pool from which to get the address of the remote host.

debug apple arp

Use the **debug apple arp** EXEC command to enable debugging of the AppleTalk Address Resolution Protocol (AARP). The **no** form of this command disables debugging output.

```
[no] debug apple arp [type number]
```

Syntax Description

type (Optional) Interface type.

number (Optional) Interface number.

Usage Guidelines

This command is helpful when you experience problems communicating with a node on the network you control (a neighbor). If the **debug apple arp** display indicates that the router is receiving AARP probes, you can assume that the problem does not reside at the physical layer.

Sample Display

Figure 2-4 shows sample **debug apple arp** output.

Figure 2-4 Sample Debug Apple ARP Output

```
router# debug apple arp

Ether0: AARP: Sent resolve for 4160.26
Ether0: AARP: Reply from 4160.26(0000.0c00.0453) for 4160.154(0000.0c00.8ea9)
Ether0: AARP: Resolved waiting request for 4160.26(0000.0c00.0453)
Ether0: AARP: Reply from 4160.19(0000.0c00.0082) for 4160.154(0000.0c00.8ea9)
Ether0: AARP: Resolved waiting request for 4160.19(0000.0c00.0082)
Ether0: AARP: Reply from 4160.19(0000.0c00.0082) for 4160.154(0000.0c00.8ea9)
```

Explanations for representative lines of output in Figure 2-4 follow.

The following line indicates that the router has requested the hardware MAC address of the host at network address 4160.26:

```
Ether0: AARP: Sent resolve for 4160.26
```

The following line indicates that the host at network address 4160.26 has replied, giving its MAC address (0000.0c00.0453). For completeness, the message also shows the network address to which the reply was sent and its hardware MAC address (also in parentheses).

```
Ether0: AARP: Reply from 4160.26(0000.0c00.0453) for 4160.154(0000.0c00.8ea9)
```

The following line indicates that the MAC address request is complete:

```
Ether0: AARP: Resolved waiting request for 4160.26(0000.0c00.0453)
```

debug apple domain

Use the **debug apple domain** EXEC command to enable debugging of the AppleTalk domain activities. The **no** form of this command disables debugging output.

[no] debug apple domain

Usage Guidelines

Use the **debug apple domain** command to observe activity for domains and subdomains. Use this command in conjunction with the **debug apple remap** command to observe interaction between remapping and domain activity. Messages are displayed when the state of a domain changes, such as creating a new domain, deleting a domain, and updating a domain.

Sample Display

Figure 2-5 shows sample **debug apple domain** output intermixed with output from the **debug apple remap** command; the two commands show related events.

Figure 2-5 Sample Debug Apple Domain Output

```
router# debug apple domain
router# debug apple remap

AT-REMAP: RemapProcess for net 30000 domain AURP Domain 1
AT-REMAP: ReshuffleRemapList for subdomain 1
AT-REMAP: Could not find a remap for cable 3000-3001
AT-DOMAIN: atdomain_DisablePort for Tunnel0
AT-DOMAIN: CleanUpDomain for domain 1 [AURP Domain 1]
AT-DOMAIN: Disabling interface Ethernet1
AT-DOMAIN: atdomain_DisablePort for Ethernet1
AT-DOMAIN: CleanUpDomain for domain 1 [AURP Domain 1]
AT-DOMAIN: CleanSubDomain for inbound subdomain 1
AT-REMAP: Remap for net 70 inbound subdomain 1 has been deleted
AT-DOMAIN: DeleteAvRemapList for inbound subdomain 1
AT-DOMAIN: DeleteRemapTable for subdomain 1
AT-DOMAIN: DeleteAvRemapList for inbound subdomain 1
AT-DOMAIN: CleanSubDomain for outbound subdomain 1
AT-DOMAIN: DeleteRemapTable for subdomain 1
AT-REMAP: RemapProcess for net 30000 domain AURP Domain 1 Remaped Net 10000
AT-REMAP: Remap for net 50 outbound subdomain 1 has been deleted
AT-DOMAIN: DeleteAvRemapList for outbound subdomain 1
AT-DOMAIN: DeleteAvRemapList for outbound subdomain 1
AT-DOMAIN: CleanUpDomain for domain 1 [AURP Domain 1]
AT-DOMAIN: CleanSubDomain for inbound subdomain 1
AT-DOMAIN: DeleteRemapTable for subdomain 1
AT-DOMAIN: DeleteAvRemapList for inbound subdomain 1
AT-DOMAIN: CleanSubDomain for outbound subdomain 1
AT-DOMAIN: DeleteRemapTable for subdomain 1
AT-DOMAIN: DeleteAvRemapList for outbound subdomain 1
```

Related Command

debug apple remap

debug apple errors

Use the **debug apple errors** EXEC command to display errors occurring in the AppleTalk network. The **no** form of this command disables debugging output.

[no] debug apple errors [*type number*]

Syntax Description

type (Optional) Interface type.

number (Optional) Interface number.

Usage Guidelines

In a stable AppleTalk network, the **debug apple errors** command produces little output.

To solve encapsulation problems, enable **debug apple errors** and **debug apple packet** together.

Sample Display

Figure 2-7 shows sample **debug apple errors** output when a router is brought up with a zone that does not agree with the zone list of other routers on the network.

Figure 2-7 Sample Debug Apple Errors Output

```
router# debug apple errors

%AT-3-ZONEDISAGREES: Ethernet0: AppleTalk port disabled; zone list incompatible with
4160.19
%AT-3-ZONEDISAGREES: Ethernet0: AppleTalk port disabled; zone list incompatible with
4160.19
%AT-3-ZONEDISAGREES: Ethernet0: AppleTalk port disabled; zone list incompatible with
4160.19
```

As Figure 2-7 suggests, a single error message indicates zone list incompatibility; this message is sent out periodically until the condition is corrected or **debug apple errors** is turned off.

Most of the other messages that **debug apple errors** can generate are obscure or indicate a serious problem with the AppleTalk network. Some of these other messages follow.

In the following message, RTMPReq, RTMPReq, ATP, AEP, ZIP, ADSP, or SNMP could replace NBP, and “llap dest not for us” could replace “wrong encapsulation”:

```
Packet discarded, src 4160.12-254,dst 4160.19-254,NBP,wrong encapsulation
```

In the following message, in addition to invalid echo packet, other possible errors are unsolicited AEP echo reply, unknown echo function, invalid ping packet, unknown ping function, and bad responder packet type.

```
Ethernet0: AppleTalk packet error; no source address available
AT: pak_reply: dubious reply creation, dst 4160.19
AT: Unable to get a buffer for reply to 4160.19

Processing error, src 4160.12-254,dst 4160.19-254,AEP, invalid echo packet
```

The **debug apple errors** command can print out additional messages when other debugging commands are also turned on. When you turn on both **debug apple errors** and **debug apple events**, the following message can be generated:

```
Proc err, src 4160.12-254,dst 4160.19-254,ZIP,NetInfo Reply format is invalid
```

In the preceding message, in addition to NetInfo Reply format is invalid, other possible errors are NetInfoReply not for me, NetInfoReply ignored, NetInfoReply for operational net ignored, NetInfoReply from invalid port, unexpected NetInfoReply ignored, cannot establish primary zone, no primary has been set up, primary zone invalid, net information mismatch, multicast mismatch, and zones disagree.

When you turn on both **debug apple errors** and **debug apple nbp**, the following message can be generated:

```
Processing error,...,NBP,NBP name invalid
```

In the preceding message, in addition to NBP name invalid, other possible errors are NBP type invalid, NBP zone invalid, not operational, error handling brq, error handling proxy, NBP fwdreq unexpected, No route to srcnet, Proxy to "*" zone, Zone "*" from extended net, No zone info for "*", and NBP zone unknown.

When you turn on both **debug apple errors** and **debug apple routing**, the following message can be generated:

```
Processing error,...,RTMPReq, unknown RTMP request
```

In the preceding message, in addition to unknown RTMP request, other possible errors are RTMP packet header bad, RTMP cable mismatch, routed RTMP data, RTMP bad tuple, and Not Req or Rsp.

debug apple events

Use the **debug apple events** EXEC command to display information about AppleTalk special events, neighbors becoming reachable/unreachable, and interfaces going up/down. Only significant events (for example, neighbor and route changes) are logged. The **no** form of this command disables debugging output.

[no] debug apple events *[type number]*

Syntax Description

<i>type</i>	(Optional) Interface type.
<i>number</i>	(Optional) Interface number.

Usage Guidelines

The **debug apple events** command is useful for solving AppleTalk network problems because it provides an overall picture of the stability of the network. In a stable network, the **debug apple events** command does not return any information. If the command generates numerous messages, those messages can indicate possible sources of the problems.

When configuring or making changes to a router or interface for AppleTalk, enable **debug apple events**. Doing so alerts you to the progress of the changes or to any errors that might result. Also use this command periodically when you suspect network problems.

The **debug apple events** command is also useful to determine whether network flapping (nodes toggling online and offline) is occurring. If flapping is excessive, look for routers that only support 254 networks.

When you enable **debug apple events**, you will see any messages that the configuration command **apple event-logging** normally displays. Turning on **debug apple events**, however, does not cause **apple event-logging** to be maintained in nonvolatile memory. Only turning on **apple event-logging** explicitly stores it in nonvolatile memory. Furthermore, if **apple event-logging** is already enabled, turning on or off **debug apple events** does not affect **apple event-logging**.

Sample Display

Figure 2-8 shows sample **debug apple events** output that describes a nonseed router coming up in discovery mode.

Figure 2-8 Sample Debug Apple Events Output—Discovery Mode State Changes

```

router# debug apple events

Discovery mode state changes
Ether0: AT: Resetting interface address filters
%AT-5-INTRESTART: Ether0: AppleTalk port restarting; protocol restarted
Ether0: AppleTalk state changed; unknown -> restarting
Ether0: AppleTalk state changed; restarting -> probing
%AT-6-ADDRUSED: Ether0: AppleTalk node up; using address 65401.148
Ether0: AppleTalk state changed; probing -> acquiring
%AT-6-ACQUIREMODE: Ether0: AT port initializing; acquiring net configuration
Ether0: AppleTalk state changed; acquiring -> restarting
Ether0: AppleTalk state changed; restarting -> line down
Ether0: AppleTalk state changed; line down -> restarting
Ether0: AppleTalk state changed; restarting -> probing
%AT-6-ADDRUSED: Ether0: AppleTalk node up; using address 4160.148
Ether0: AppleTalk state changed; probing -> acquiring
%AT-6-ACQUIREMODE: Ether0: AT port initializing; acquiring net configuration
Ether0: AppleTalk state changed; acquiring -> requesting zones
Ether0: AT: Resetting interface address filters
%AT-5-INTRESTART: Ether0: AppleTalk port restarting; protocol restarted
Ether0: AppleTalk state changed; requesting zones -> verifying
AT: Sent GetNetInfo request broadcast on Ethernet0
Ether0: AppleTalk state changed; verifying -> checking zones
Ether0: AppleTalk state changed; checking zones -> operational

```

As Figure 2-8 shows, the **debug apple events** command is useful in tracking the discovery mode state changes through which an interface progresses. When no problems are encountered, the state changes progress as follows:

- 1 Line down
- 2 Restarting
- 3 Probing (for its own address [node ID] using AARP)
- 4 Acquiring (sending out GetNetInfo requests)
- 5 Requesting zones (the list of zones for its cable)
- 6 Verifying (that the router's configuration is correct. If not, a port configuration mismatch is declared.)
- 7 Checking zones (to make sure its list of zones is correct)
- 8 Operational (participating in routing)

Explanations for individual lines of output in Figure 2-8 follow.

The following message indicates that a port is set. In this case, the zone multicast address is being reset:

```
Ether0: AT: Resetting interface address filters
```

The following messages indicate that the router is changing to restarting mode:

```
%AT-5-INTRESTART: Ether0: AppleTalk port restarting; protocol restarted
Ether0: AppleTalk state changed; unknown -> restarting
```

The following message indicates that the router is probing in the startup range of network numbers (65280-65534) to discover its network number:

```
Ether0: AppleTalk state changed; restarting -> probing
```

The following message indicates that the router is enabled as a nonrouting node using a provisional network number within its startup range of network numbers. This type of message only appears if the network address the router will use differs from its configured address. This is always the case for a discovery-enabled router; it is rarely the case for a nondiscovery-enabled router.

```
%AT-6-ADDRUSED: Ether0: AppleTalk node up; using address 65401.148
```

The following messages indicate that the router is sending out GetNetInfo requests to discover the default zone name and the actual network number range in which its network number can be chosen:

```
Ether0: AppleTalk state changed; probing -> acquiring
%AT-6-ACQUIREMODE: Ether0: AT port initializing; acquiring net configuration
```

Now that the router has acquired the cable configuration information, the following message indicates that it restarts using that information:

```
Ether0: AppleTalk state changed; acquiring -> restarting
```

The following messages indicate that the router is probing for its actual network address:

```
Ether0: AppleTalk state changed; restarting -> line down
Ether0: AppleTalk state changed; line down -> restarting
Ether0: AppleTalk state changed; restarting -> probing
```

The following message indicates that the router has found an actual network address to use:

```
%AT-6-ADDRUSED: Ether0: AppleTalk node up; using address 4160.148
```

The following messages indicate that the router is sending out GetNetInfo requests to verify the default zone name and the actual network number range from which its network number can be chosen:

```
Ether0: AppleTalk state changed; probing -> acquiring
%AT-6-ACQUIREMODE: Ether0: AT port initializing; acquiring net configuration
```

The following message indicates that the router is requesting the list of zones for its cable:

```
Ether0: AppleTalk state changed; acquiring -> requesting zones
```

The following messages indicate that the router is sending out GetNetInfo requests to make sure its understanding of the configuration is correct:

```
Ether0: AppleTalk state changed; requesting zones -> verifying
AT: Sent GetNetInfo request broadcast on Ethernet0
```

The following message indicates that the router is rechecking its list of zones for its cable:

```
Ether0: AppleTalk state changed; verifying -> checking zones
```

The following message indicates that the router is now fully operational as a routing node and can begin routing:

```
Ether0: AppleTalk state changed; checking zones -> operational
```

Figure 2-9 shows sample **debug apple events** output that describes a nondiscovery-enabled router coming up when no other router is on the wire.

Figure 2-9 Sample Debug Apple Events Output—Seed Coming Up by Itself

Indicates a nondiscovery-enabled router with no other router on the wire

```
router# debug apple events

Ethernet1: AT: Resetting interface address filters
%AT-5-INTRESTART: Ethernet1: AppleTalk port restarting; protocol restarted
Ethernet1: AppleTalk state changed; unknown -> restarting
Ethernet1: AppleTalk state changed; restarting -> probing
%AT-6-ADDRUSED: Ethernet1: AppleTalk node up; using address 4165.204
Ethernet1: AppleTalk state changed; probing -> verifying
AT: Sent GetNetInfo request broadcast on Ethernet1
Ethernet1: AppleTalk state changed; verifying -> operational
%AT-6-ONLYROUTER: Ethernet1: AppleTalk port enabled; no neighbors found
```

S2543

As Figure 2-9 shows, a nondiscovery-enabled router can come up when no other router is on the wire; however, it must assume that its configuration (if accurate syntactically) is correct, because no other router can verify it. Notice that the last line in Figure 2-9 indicates this situation.

Figure 2-10 shows sample **debug apple events** output that describes a discovery-enabled router coming up when there is no seed router on the wire.

Figure 2-10 Sample Debug Apple Events Output—Nonseed with No Seed

```
router# debug apple events

Ether0: AT: Resetting interface address filters
%AT-5-INTRESTART: Ether0: AppleTalk port restarting; protocol restarted
Ether0: AppleTalk state changed; unknown -> restarting
Ether0: AppleTalk state changed; restarting -> probing
%AT-6-ADDRUSED: Ether0: AppleTalk node up; using address 65401.148
Ether0: AppleTalk state changed; probing -> acquiring
AT: Sent GetNetInfo request broadcast on Ether0
AT: Sent GetNetInfo request broadcast on Ether0
AT: Sent GetNetInfo request broadcast on Ether0
AT: Sent GetNetInfo request broadcast on Ether0
AT: Sent GetNetInfo request broadcast on Ether0
```

As Figure 2-10 shows, when you attempt to bring up a nonseed router without a seed router on the wire, it never becomes operational; instead, it hangs in the acquiring mode and continues to send out periodic GetNetInfo requests.

Figure 2-11 shows sample **debug apple events** output when a nondiscovery-enabled router is brought up on an AppleTalk internetwork that is in compatibility mode (set up to accommodate extended as well as nonextended AppleTalk) and the router has violated internetwork compatibility.

Figure 2-11 Sample Debug Apple Events Output—Compatibility Conflict

Indicates configuration mismatch

```
router# debug apple events

E0: AT: Resetting interface address filters
%AT-5-INTRESTART: E0: AppleTalk port restarting; protocol restarted
E0: AppleTalk state changed; restarting -> probing
%AT-6-ADDRUSED: E0: AppleTalk node up; using address 41.19
E0: AppleTalk state changed; probing -> verifying
AT: Sent GetNetInfo request broadcast on Ethernet0
%AT-3-ZONEDISAGREES: E0: AT port disabled; zone list incompatible with 41.19
AT: Config error for E0, primary zone invalid
E0: AppleTalk state changed; verifying -> config mismatch
```

S2545

The three configuration command lines that follow indicate the part of the router's configuration that caused the configuration mismatch shown in Figure 2-11:

```
lestat(config)#int e 0
lestat(config-if)#apple cab 41-41
lestat(config-if)#apple zone Marketing
```

The router shown in Figure 2-11 had been configured with a cable range of 41-41 instead of 40-40, which would have been accurate. Additionally, the zone name was configured incorrectly; it should have been "Marketing," rather than being misspelled as "Marketing."

debug apple nbp

Use the **debug apple nbp** EXEC command to display debugging output from the Name Binding Protocol (NBP) routines. The **no** form of this command disables debugging output.

[no] debug apple nbp [*type number*]

Syntax Description

type (Optional) Interface type.

number (Optional) Interface number.

Usage Guidelines

To determine whether the router is receiving NBP lookups from a node on the AppleTalk network, enable **debug apple nbp** at each node between the router and the node in question to determine where the problem lies.

Note Because the **debug apple nbp** command can generate many messages, use it only when the router's CPU utilization is less than 50 percent.

Sample Display

Figure 2-12 shows sample **debug apple nbp** output.

Figure 2-12 Sample Debug Apple NBP Output

```
router# debug apple nbp

AT: NBP ctrl = LkUp, ntuples = 1, id = 77
AT: 4160.19, skt 2, enum 0, name: =:ciscoRouter@Low End SW Lab
AT: LkUp =:ciscoRouter@Low End SW Lab

AT: NBP ctrl = LkUp-Reply, ntuples = 1, id = 77
AT: 4160.154, skt 254, enum 1, name: lestat.Ether0:ciscoRouter@Low End SW Lab

AT: NBP ctrl = LkUp, ntuples = 1, id = 78
AT: 4160.19, skt 2, enum 0, name: =:IPADDRESS@Low End SW Lab
AT: NBP ctrl = LkUp, ntuples = 1, id = 79
AT: 4160.19, skt 2, enum 0, name: =:IPGATEWAY@Low End SW Lab
AT: NBP ctrl = LkUp, ntuples = 1, id = 83
AT: 4160.19, skt 2, enum 0, name: =:ciscoRouter@Low End SW Lab
AT: LkUp =:ciscoRouter@Low End SW Lab

AT: NBP ctrl = LkUp, ntuples = 1, id = 84
AT: 4160.19, skt 2, enum 0, name: =:IPADDRESS@Low End SW Lab

AT: NBP ctrl = LkUp, ntuples = 1, id = 85
AT: 4160.19, skt 2, enum 0, name: =:IPGATEWAY@Low End SW Lab
AT: NBP ctrl = LkUp, ntuples = 1, id = 85
AT: 4160.19, skt 2, enum 0, name: =:IPGATEWAY@Low End SW Lab
```

The first three lines in Figure 2-12 describe an NBP lookup request:

```
AT: NBP ctrl = LkUp, ntuples = 1, id = 77
AT: 4160.19, skt 2, enum 0, name: =:ciscoRouter@Low End SW Lab
AT: LkUp =:ciscoRouter@Low End SW Lab
```

Table 2-3 describes the fields in the first line of output shown in Figure 2-12.

Table 2-3 Debug Apple NBP Field Descriptions—Part 1

Field	Description
AT: NBP	Indicates that this message describes an AppleTalk NBP packet.
ctrl = LkUp	Identifies the type of NBP packet. Possible values include LkUp—NBP lookup request. LkUp-Reply—NBP lookup reply.
ntuples = 1	Indicates the number of name-address pairs in the lookup request packet. Range: 1-31 tuples.
id = 77	Identifies an NBP lookup request value.

Table 2-4 describes the fields in the second line of output shown in Figure 2-12.

Table 2-4 Debug Apple NBP Field Descriptions—Part 2

Field	Description
AT:	Indicates that this message describes an AppleTalk packet.
4160.19	Indicates the network address of the requester.
skt 2	Indicates the internet socket address of the requester. The responder will send the NBP lookup reply to this socket address.
enum 0	Indicates the enumerator field. Used to identify multiple names registered on a single socket. Each tuple is assigned its own enumerator, incrementing from 0 for the first tuple.
name: =:ciscoRouter@Low End SW Lab	Indicates the entity name for which a network address has been requested. The AppleTalk entity name includes three components: Object (in this case, a wildcard character (=), indicating that the requester is requesting name-address pairs for all objects of the specified type in the specified zone) Type (in this case, ciscoRouter) Zone (in this case, Low End SW Lab)

The third line in Figure 2-12 essentially reiterates the information in the two lines above it, indicating that a lookup request has been made regarding name-address pairs for all objects of the ciscoRouter type in the Low End SW Lab zone.

Because the router is defined as an object of type ciscoRouter in zone Low End SW Lab, the router sends an NBP lookup reply in response to this NBP lookup request. The following two lines of output from Figure 2-12 show the router’s response:

```
AT: NBP ctrl = LkUp-Reply, ntuples = 1, id = 77
AT: 4160.154, skt 254, enum 1, name: lestat.Ether0:ciscoRouter@Low End SW Lab
```

In the first line, ctrl = LkUp-Reply identifies this NBP packet as an NBP lookup request. The same value in the id field (id = 77) associates this lookup reply with the previous lookup request. The second line indicates that the network address associated with the router's entity name (lestat.Ether0:ciscoRouter@Low End SW Lab) is 4160.154. The fact that no other entity name/network address is listed indicates that the responder only knows about itself as an object of type ciscoRouter in zone Low End SW Lab.

debug apple packet

Use the **debug apple packet** EXEC command to display per-packet debugging output. The output reports information online when a packet is received or a transmit is attempted. The **no** form of this command disables debugging output.

[no] debug apple packet [*type number*]

Syntax Description

type (Optional) Interface type.

number (Optional) Interface number.

Usage Guidelines

With this command, you can monitor the types of packets being slow switched. It displays at least one line of debugging output per AppleTalk packet processed.

When invoked in conjunction with the **debug apple routing**, **debug apple zip**, and **debug apple nbp** commands, the **debug apple packet** command adds protocol processing information in addition to generic packet details. It also reports successful completion or failure information.

When invoked in conjunction with the **debug apple errors** command, the **debug apple packet** command reports packet-level problems, such as those concerning encapsulation.

Note Because the **debug apple packet** command can generate many messages, use it only when the router's CPU utilization is less than 50 percent.

Sample Display

Figure 2-13 shows sample **debug apple packet** output.

Figure 2-13 Sample Debug Apple Packet Output

```
router# debug apple packet

Ether0: AppleTalk packet: enctype SNAP, size 60, encaps000000000000000000000000
AT: src=Ethernet0:4160.47, dst=4160-4160, size=10, 2 rtes, RTMP pkt sent
AT: ZIP Extended reply rcvd from 4160.19
AT: ZIP Extended reply rcvd from 4160.19
AT: src=Ethernet0:4160.47, dst=4160-4160, size=10, 2 rtes, RTMP pkt sent
Ether0: AppleTalk packet: enctype SNAP, size 60, encaps000000000000000000000000
Ether0: AppleTalk packet: enctype SNAP, size 60, encaps000000000000000000000000
```

Table 2-5 describes the fields in the first line of output shown in Figure 2-13.

Table 2-5 Debug Apple Packet Field Descriptions—Part 1

Field	Description
Ether0:	Name of the interface through which the router received the packet
AppleTalk packet	Indication that this is an AppleTalk packet
encype SNAP	Encapsulation type for the packet
size 60	Size of the packet (in bytes)
encaps000000000000000000000000	Encapsulation

Table 2-6 describes the fields in the second line of output shown in Figure 2-13.

Table 2-6 Debug Apple Packet Field Descriptions—Part 2

Field	Description
AT:	Indication that this is an AppleTalk packet
src = Ethernet0:4160.47	Name of the interface sending the packet and its AppleTalk address
dst = 4160-4160	Cable range of the packet's destination
size = 10	Size of the packet (in bytes)
2 rtes	Indication that two routes in the routing table link these two addresses
RTMP pkt sent	The type of packet sent

The third line in Figure 2-13 indicates the type of packet received and its source AppleTalk address. This message is repeated in the fourth line because AppleTalk hosts can send multiple replies to a given GetNetInfo request.

debug apple remap

Use the **debug apple remap** EXEC command to enable debugging of the AppleTalk remap activities. The **no** form of this command disables debugging output.

[no] debug apple remap

Usage Guidelines

Use the **debug apple remap** command with the **debug apple domain** command to observe activity between domains and subdomains. Messages from **debug apple remap** are displayed when a particular remapping function occurs, such as creating remaps or deleting remaps.

Sample Display

Figure 2-14 shows sample **debug apple remap** output intermixed with output from the **debug apple domain** command; the two commands show related events.

Figure 2-14 Sample Debug Apple Remap and Domain Output

```
router# debug apple remap
router# debug apple domain

AT-REMAP: RemapProcess for net 30000 domain AURP Domain 1
AT-REMAP: ReshuffleRemapList for subdomain 1
AT-REMAP: Could not find a remap for cable 3000-3001
AT-DOMAIN: atdomain_DisablePort for Tunnel0
AT-DOMAIN: CleanUpDomain for domain 1 [AURP Domain 1]
AT-DOMAIN: Disabling interface Ethernet1
AT-DOMAIN: atdomain_DisablePort for Ethernet1
AT-DOMAIN: CleanUpDomain for domain 1 [AURP Domain 1]
AT-DOMAIN: CleanSubDomain for inbound subdomain 1
AT-REMAP: Remap for net 70 inbound subdomain 1 has been deleted
AT-DOMAIN: DeleteAvRemapList for inbound subdomain 1
AT-DOMAIN: DeleteRemapTable for subdomain 1
AT-DOMAIN: DeleteAvRemapList for inbound subdomain 1
AT-DOMAIN: CleanSubDomain for outbound subdomain 1
AT-DOMAIN: DeleteRemapTable for subdomain 1
AT-REMAP: RemapProcess for net 30000 domain AURP Domain 1 Remaped Net 10000
AT-REMAP: Remap for net 50 outbound subdomain 1 has been deleted
AT-DOMAIN: DeleteAvRemapList for outbound subdomain 1
AT-DOMAIN: DeleteAvRemapList for outbound subdomain 1
AT-DOMAIN: CleanUpDomain for domain 1 [AURP Domain 1]
AT-DOMAIN: CleanSubDomain for inbound subdomain 1
AT-DOMAIN: DeleteRemapTable for subdomain 1
AT-DOMAIN: DeleteAvRemapList for inbound subdomain 1
AT-DOMAIN: CleanSubDomain for outbound subdomain 1
AT-DOMAIN: DeleteRemapTable for subdomain 1
AT-DOMAIN: DeleteAvRemapList for outbound subdomain 1
```

Related Command

debug apple domain

debug apple routing

Use the **debug apple routing** EXEC command to enable debugging output from the Routing Table Maintenance Protocol (RTMP) routines. The **no** form of this command disables debugging output.

```
[no] debug apple routing [type number]
```

Syntax Description

type (Optional) Interface type.

number (Optional) Interface number.

Usage Guidelines

This command can be used to monitor acquisition of routes, aging of routing table entries, and advertisement of known routes. It also reports conflicting network numbers on the same network if the network is misconfigured.

Note Because the **debug apple routing** command can generate many messages, use it only when router CPU utilization is less than 50 percent.

Sample Display

Figure 2-15 shows sample **debug apple routing** output.

Figure 2-15 Sample Debug Apple Routing Output

```
router# debug apple routing

AT: src=Ethernet0:4160.41, dst=4160-4160, size=19, 2 rtes, RTMP pkt sent
AT: src=Ethernet1:41069.25, dst=41069, size=427, 96 rtes, RTMP pkt sent
AT: src=Ethernet2:4161.23, dst=4161-4161, size=427, 96 rtes, RTMP pkt sent
AT: Route ager starting (97 routes)
AT: Route ager finished (97 routes)
AT: RTMP from 4160.19 (new 0,old 94,bad 0,ign 0, dwn 0)
AT: RTMP from 4160.250 (new 0,old 0,bad 0,ign 2, dwn 0)
AT: RTMP from 4161.236 (new 0,old 94,bad 0,ign 1, dwn 0)
AT: src=Ethernet0:4160.41, dst=4160-4160, size=19, 2 rtes, RTMP pkt sent
```

Explanations for representative lines of the **debug apple routing** output in Figure 2-15 follow.

Table 2-7 describes the fields in the first line of sample **debug apple routing** output.

Table 2-7 Debug Apple Routing Field Descriptions—Part 1

Field	Description
AT:	Indicates that this is AppleTalk debugging output
src = Ethernet0:4160.41	Indicates the source router interface and network address for the RTMP update packet
dst = 4160-4160	Indicates the destination network address for the RTMP update packet

Table 2-7 Debug Apple Routing Field Descriptions—Part 1 (Continued)

Field	Description
size = 19	Shows the size of this RTMP packet (in bytes)
2 rtes	Indicates that this RTMP update packet includes information on two routes
RTMP pkt sent	Indicates that this type of message describes an RTMP update packet that the router has sent (rather than one that it has received)

The following two messages indicate that the ager has started and finished the aging process for the routing table and that this table contains 97 entries.

```
AT: Route ager starting (97 routes)
AT: Route ager finished (97 routes)
```

Table 2-8 describes the fields in the following line of **debug apple routing** output.

```
AT: RTMP from 4160.19 (new 0,old 94,bad 0,ign 0, dwn 0)
```

Table 2-8 Debug Apple Routing Field Descriptions—Part 2

Field	Description
AT:	Indicates that this is AppleTalk debugging output
RTMP from 4160.19	Indicates the source address of the RTMP update the router received
new 0	Shows the number of routes in this RTMP update packet that the router did not already know about
old 94	Shows the number of routes in this RTMP update packet that the router already knew about
bad 0	Shows the number of routes the other router indicates have gone bad
ign 0	Shows the number of routes the other router ignores
dwn 0	Shows the number of poisoned tuples included in this packet

debug apple zip

Use the **debug apple zip** EXEC command to display debugging output from the Zone Information Protocol (ZIP) routines. The **no** form of this command disables debugging output.

[no] debug apple zip [*type number*]

Syntax Description

type (Optional) Interface type.

number (Optional) Interface number.

Usage Guidelines

This command reports significant events such as the discovery of new zones and zone list queries. It generates information similar to that generated by **debug apple routing**, but generates it for ZIP packets instead of RTMP packets.

You can use the **debug apple zip** command to determine whether a ZIP storm is taking place in the AppleTalk network. You can detect the existence of a ZIP storm when you see that no router on a cable has the zone name corresponding to a network number that all the routers have in their routing tables.

Sample Display

Figure 2-16 shows sample **debug apple zip** output.

Figure 2-16 Sample Debug Apple ZIP Output

```
router# debug apple zip

AT: Sent GetNetInfo request broadcast on Ether0
AT: Recvd ZIP cmd 6 from 4160.19-6
AT: 3 query packets sent to neighbor 4160.19
AT: 1 zones for 31902, ZIP XReply, src 4160.19
AT: net 31902, zonelen 10, name US-Florida
```

Explanations of the lines of output shown in Figure 2-16 follow.

The first line indicates that the router has received an RTMP update that includes a new network number and is now requesting zone information:

```
AT: Sent GetNetInfo request broadcast on Ether0
```

The second line indicates that the neighbor at address 4160.19 replies to the zone request with a default zone:

```
AT: Recvd ZIP cmd 6 from 4160.19-6
```

The third line indicates that the router responds with three queries to the neighbor at network address 4160.19 for other zones on the network:

```
AT: 3 query packets sent to neighbor 4160.19
```

The fourth line indicates that the neighbor at network address 4160.19 responds with a ZIP extended reply, indicating that one zone has been assigned to network 31902:

```
AT: 1 zones for 31902, ZIP XReply, src 4160.19
```

The fifth line indicates that the router responds that the zone name of network 31902 is US-Florida, and the zone length of that zone name is 10:

```
AT: net 31902, zonelen 10, name US-Florida
```

debug appn all

Use the **debug appn all** EXEC command to turn on all possible debugging messages for Advanced Peer-to-Peer Networking (APPN). The **no** form of this command disables debugging output.

[no] debug appn all

Note Refer to the other forms of the **debug appn** command to enable specific debug output selectively.

Usage Guidelines

This command shows all APPN events. Use other forms of the **debug appn** command to display specific types of events.

Note Because the **debug appn all** command can generate many messages and alter timing in the network node, use it only when instructed by authorized support personnel.



Caution Debugging output takes priority over other network traffic. The **debug appn all** command generates more output than any other **debug appn** command and can alter timing in the network node. This command can severely diminish router performance or even render it unusable. In virtually all cases, it is best to use specific **debug appn** commands.

Related Commands

debug appn cs
debug appn ds
debug appn ms
debug appn nof
debug appn pc
debug appn ps
debug appn scm
debug appn ss
debug appn trs

debug appn cs

Use the **debug appn cs** EXEC command to display APPN Configuration Services (CS) component activity. The **no** form of this command disables debugging output.

[no] debug appn cs

Usage Guidelines

The Configuration Services (CS) component is responsible for defining link stations, ports, and connection networks. It is responsible for the activation and deactivation of ports and link stations and handles status queries for these resources.

Sample Display

Figure 2-17 shows sample **debug appn cs** output. In this example a link station is being stopped.

Figure 2-17 Sample Debug APPN CS Output

```
router# debug appn cs

Turned on event 008000FF

router# appn stop link PATTY

APPN: ----- CS ----- Deq STOP_LS message
APPN: ----- CS ----- FSM LS: 75 17 5 8
APPN: ----- CS ----- Sending DEACTIVATE_AS - station PATTY
APPN: ----- CS ----- deactivate_as_p->ips_header.lpid = A80A60
APPN: ----- CS ----- deactivate_as_p->ips_header.lpid = A80A60
APPN: ----- CS ----- Sending DESTROY_TG to PC - station PATTY - lpid=A80A60
APPN: ----- CS ----- Deq DESTROY_TG - station PATTY
APPN: ----- CS ----- FSM LS: 22 27 8 0
APPN: ----- CS ----- Sending TG update for LS PATTY to TRS
APPN: ----- CS ----- ENTERING XID_PROCESSING: 4
%APPN-6-APPNSENDMSG: Link Station PATTY stopped
```

Table 2-9 shows describes the fields and messages shown in Figure 2-17.

Table 2-9 Debug APPN CS Field Descriptions

Field	Description
APPN	APPN debugging output.
CS	Configuration Services component output.
Deq	CS received a message from another component.
FSM LS	The link station finite state machine is being referenced.
Sending	CS is sending a message to another component.

Related Command

debug appn all

debug appn ds

Use the **debug appn ds** EXEC command to display debugging information on APPN Directory Services (DS) component activity. The **no** form of this command disables debugging output.

[no] debug appn ds

Usage Guidelines

The Directory Services (DS) component manages searches for resources in the APPN network. DS is also responsible for registration of resources within the network.

Sample Display

Figure 2-18 shows sample **debug appn ds** output. In this example a search has been received.

Figure 2-18 Sample Debug APPN DS Output

```
router# debug appn ds
Turned on event 080000FF
APPN: NEWDS: LS: search from: NETA.PATTY
APPN: NEWDS: pcid: DD3321E8B5667111
APPN: NEWDS: Invoking FSM NNSolu
APPN: NEWDS: LSfsm_NNSolu: 00A67AA0 pcid: DD3321E8B5667111 row: 0 col: 0 inp: 80200000
APPN: NEWDS: LSfsm_parent: 00A89940 row: 0 col: 0 inp: 80000000
APPN: NEWDS: Rcvd a LMRQ
APPN: NEWDS: LSfsm_NNSolu: 00A67AA0 pcid: DD3321E8B5667111 row: 12 col: 1 inp: 40000000
APPN: NEWDS: LSfsm_parent: 00A89940 row: 8 col: 1 inp: 40000000
APPN: NEWDS: LSfsm_child: 00A89BE8 row: 0 col: 0 inp: 80000080
APPN: NEWDS: PQenq REQUEST_ROUTE(RQ) to TRS
APPN: NEWDS: LSfsm_child: 00A8A1C0 row: 1 col: 0 inp: 80000008
APPN: NEWDS: LSfsm_NNSolu: 00A67AA0 pcid: DD3321E8B5667111 row: 5 col: 1 inp: 80C04000
APPN: NEWDS: LSfsm_child: 00A8A1C0 row: 7 col: 1 inp: 80844008
APPN: NEWDS: Rcvd a LMRQ
APPN: NEWDS: LSfsm_NNSolu: 00A67AA0 pcid: DD3321E8B5667111 row: 16 col: 6 inp: 40800000
APPN: NEWDS: LSfsm_child: 00A8A1C0 row: 14 col: 5 inp: 40800000
APPN: NEWDS: LSfsm_parent: 00A89940 row: 3 col: 1 inp: 80840000
APPN: NEWDS: send locate to node: NETA.PATTY
```

Table 2-10 provides explanations for fields in the **debug appn ds** output shown in Figure 2-18.

Table 2-10 Debug APPN DS Field Descriptions

Field	Description
APPN	APPN debugging output.
NEWDS	Directory Services component output.
search from	A locate was received from NETA.PATTY.
LSfsm_	The Locate Search finite state machine is being referenced.
PQenq	A message was sent to another component.
Rcvd	A message was received from another component.
send locate	A locate will be sent to NETA.PATTY.

Related Command

debug appn all

debug appn ms

Use the **debug appn ms** EXEC command to display debugging information on APPN Management Services (MS) component activity. The **no** form of this command disables debugging output.

[no] debug appn ms

Usage Guidelines

The Management Services (MS) component is responsible for generating, sending, and forwarding network management information in the form of traps and alerts to a network management focal point, such as Netview, in the APPN network.

Sample Display

Figure 2-19 shows sample **debug appn ms** output. In this example an error occurred that caused an alert to be generated.

Figure 2-19 Sample Debug APPN MS Output

```
router# debug appn ms

APPN: ----- MSS00 ---- Deq ALERT_MSU msg
APPN: --- MSP70 --- ALERT MV FROM APPN WITH VALID LGTH
APPN: --- MSCPL --- Find Active FP
APPN: --- MSP30 --- Entering Build MS Transport
APPN: --- MSP31 --- Entering Building Routing Info.
APPN: --- MSP34 --- Entering Build GDS
APPN: --- MSP32 --- Entering Building UOW correlator
APPN: --- MSP34 --- Entering Build GDS
APPN: --- MSP30 --- Building GDS 0x1310
APPN: --- MSP30 --- Building MS Transport
APPN: --- MSP72 --- ACTIVE FP NOT FOUND, SAVE ONLY
APPN: --- MSUTL --- UOW <= 60, ALL COPIED in extract_uow
APPN: --- MSCAT --- by enq_cached_ms QUEUE SIZE OF QUEUE after enq 4
```

Table 2-11 describes fields in the **debug appn ms** output shown in Figure 2-19.

Table 2-11 Debug APPN MS Output Field Descriptions

Field	Description
APPN	Indicates that this is APPN debugging output.
MSP	Indicates that this is MS component output.

Related Command

debug appn all

debug appn nof

Use the **debug appn nof** EXEC command to display debugging information on APPN Node Operator Facility (NOF) component activity. The **no** form of this command disables debugging output.

[no] debug appn nof

Usage Guidelines

The Node Operator Facility (NOF) component is responsible for processing commands entered by the user such as start, stop, show, and configuration commands. NOF forwards these commands to the proper component and wait for the response.

Sample Display

Figure 2-20 shows sample **debug appn nof** output. In this example an APPN connection network is being defined.

Figure 2-20 Sample Debug APPN NOF Output

```
router# debug appn nof
Turned on event 010000FF

router# config term

Enter configuration commands, one per line. End with CNTL/Z.
router(config)#appn connection-network NETA.CISCO
router(config-appn-cn)#port TR0
router(config-appn-cn)#complete
router(config)#

APPN: ----- NOF ----- Define Connection Network Verb Received
APPN: ----- NOF ----- send define_cn_t ips to cs
APPN: ----- NOF ----- waiting for define_cn rsp from cs
router(config)#
```

Table 2-12 describes fields in the **debug appn nof** output shown in Figure 2-20.

Table 2-12 Debug APPN NOF Field Descriptions

Field	Description
APPN	APPN debugging output.
NOF	NOF component output.
Received	A configuration command was entered.
send	A message was sent to CS.
waiting	A response was expected from CS.

Related Command

debug appn all

debug appn pc

Use the **debug appn pc** EXEC command to display debugging information on APPN Path Control (PC) component activity. The **no** form of this command disables debugging output.

[no] debug appn pc

Usage Guidelines

The Path Control (PC) component is responsible for passing Message Units (MUs) between the Data Link Control (DLC) layer and other APPN components. PC implements transmission priority by passing higher priority MUs to the DLC before lower priority MUs.

Sample Display

Figure 2-21 shows sample **debug appn pc** output. In this example a MU is received from the network.

Figure 2-21 Sample Debug APPN PC Output

```
router# debug appn pc

Turned on event 040000FF
APPN: ----- PC-----PC Deq REMOTE msg variant_name 2251
APPN: --PC-- mu received to PC lpid: A80AEC
APPN: --PC-- mu received from p_cep_id: 67C6F8
APPN: ----- PC-----PC Deq LSA_IPS from DLC
APPN: --PCX dequeued a DATA.IND
APPN: --- PC processing DL_DATA.ind
APPN: --PC-- mu_error_checker with no error, calling frr
APPN: --PC-- calling frr for packet received on LFSID: 1 2 3
APPN: ----- PC-----PC is sending MU to SC A90396
APPN: ----- SC-----send mu: A90396, rpc: 0, nws: 7, rh.b1: 90
APPN: SC: Send mu.snf: 8, th.b0: 2E, rh.b1: 90, dcf: 8
```

Table 2-13 describes fields in the **debug appn pc** output shown in Figure 2-21.

Table 2-13 Debug APPN PC Field Descriptions

Field	Description
APPN	APPN debugging output.
PC	PC component output.
Deq REMOTE	A message was received from the network.
mu received	The message is a MU.
DATA.IND	The MU contains data.
sending MU	The MU is session traffic for an ISR session. The MU is forwarded to the Session Connector component for routing.

Related Command

debug appn all

debug appn ps

Use the **debug appn ps** EXEC command to display debugging information on APPN Presentation Services (PS) component activity. The **no** form of this command disables debugging output.

[no] debug appn ps

Usage Guidelines

The Presentation Services (PS) component is responsible for managing the Transaction Programs (TPs) used by APPN. TPs are used for sending and receiving searches, receiving resource registration, and sending and receiving topology updates.

Sample Display

Figure 2-22 shows sample **debug appn ps** output. In this example a CP capabilities exchange is in progress.

Figure 2-22 Sample Debug APPN PS Output

```
router# debug appn ps

Turned on event 200000FF
APPN: ---- CCA --- CP_CAPABILITIES_TP has started
APPN: ---- CCA --- About to wait for Partner to send CP_CAP
APPN: ---- CCA --- Partner LU name: NETA.PATTY
APPN: ---- CCA --- Mode Name: CPSVCMG
APPN: ---- CCA --- CGID: 78
APPN: ---- CCA --- About to send cp_cp_session_act to SS
APPN: ---- CCA --- Waiting for cp_cp_session_act_rsp from SS
APPN: ---- CCA --- Received cp_cp_session_act_rsp from SS
APPN: ---- CCA --- About to send CP_CAP to partner
APPN: ---- CCA --- Send to partner completed with rc=0, 0
APPN: ---- RCA --- Allocating conversation
APPN: ---- RCA --- Sending CP_CAPABILITIES
APPN: ---- RCA --- Getting conversation attributes
APPN: ---- RCA --- Waiting for partner to send CP_CAPABILITIES
APPN: ---- RCA --- Normal processing complete with cgid = 82
APPN: ---- RCA --- Deallocating CP_Capabilities conversation
```

Table 2-14 describes fields in the **debug appn ps** output shown in Figure 2-22.

Table 2-14 Debug APPN PS Field Descriptions

Field	Description
APPN	APPN debugging output.
CCA	CP Capabilities TP output.
RCA	Receive CP Capabilities TP output.

Related Command

debug appn all

debug appn scm

Use the **debug appn scm** EXEC command to display debugging information on APPN Session Connector Manager (SCM) component activity. The **no** form of this command disables debugging output.

[no] debug appn scm

Usage Guidelines

The Session Connector Manager (SCM) component is responsible for the activation and deactivation the local resources that route an intermediate session through the router.

Sample Display

Figure 2-23 shows sample **debug appn scm** output. In this example an intermediate session traffic is being routed.

Figure 2-23 Sample Debug APPN SCM Output

```
router# debug appn scm

Turned on event 020000FF
router#
APPN: ----- SCM-----SCM Deq a MU
APPN: ----- SCM-----SCM send ISR_INIT to SSI
APPN: ----- SCM----- (i05) Enter compare_fqpcid()
APPN: ----- SCM-----Adding new session_info table entry. addr=A93160
APPN: ----- SCM-----SCM Deq ISR_CINIT message
APPN: ----- SCM----- (i05) Enter compare_fqpcid()
APPN: ----- SCM-----SCM sends ASSIGN_LFSID to ASM
APPN: ----- SCM-----SCM Rcvd sync ASSIGN_LFSID from ASM
APPN: ----- SCM-----SCM PQenq a MU to ASM
APPN: ----- SCM-----SCM Deq a MU
APPN: ----- SCM----- (i05) Enter compare_fqpcid()
APPN: ----- SCM-----SCM PQenq BIND rsp to ASM
```

Table 2-15 describes fields in the **debug appn ps** output shown in Figure 2-23.

Table 2-15 Debug APPN SCM Field Descriptions

Field	Description
APPN	APPN debugging output.
SCM	SCM component output.

Related Command

debug appn all

debug appn ss

Use the **debug appn ss** EXEC command to display session services (SS) events. The **no** form of this command disables debugging output.

[no] debug appn ss

Usage Guidelines

The Session Services (SS) component generates unique session identifiers, activates and deactivates control point-to-control point (CP-CP) sessions, and assists LUs in initiating and activating LU-LU sessions.

Sample Display

Figure 2-24 shows sample **debug appn ss** output. In this example CP-CP sessions between the router and another node are being activated.

Figure 2-24 Sample Debug APPN SS Output

```
router# debug appn ss

Turned on event 100000FF
APPN: ----- SS ----- Deq ADJACENT_CP_CONTACTED message
APPN: ----- SS ----- Deq SESSST_SIGNAL message
APPN: ----- SS ----- Deq CP_CP_SESSION_ACT message
APPN: Sending ADJACENT_NN_1015 to SCM, adj_node_p=A6B980,cp_name=NETA.PATTY
APPN: ----- SS ----- Sending REQUEST_LAST_FRSN message to TRS
APPN: ----- SS ----- Receiving REQUEST_LAST_FRSN_RSP from TRS
APPN: ----- SS ----- Sending ACTIVE CP_STATUS CONLOSER message to DS
APPN: ----- SS ----- Sending ACTIVE CP_STATUS CONLOSER message to MS
APPN: ----- SS ----- Sending ACTIVE CP_STATUS CONLOSER message to TRS
APPN: ----- SS ----- Sending CP_CP_SESSION_ACT_RSP message to CCA TP
APPN: ----- SS ----- Sending PENDING_ACTIVE CP_STATUS CONWINNER message to DS
APPN: ----- SS ----- Sending REQUEST_LAST_FRSN message to TRS
APPN: ----- SS ----- Receiving REQUEST_LAST_FRSN_RSP from TRS
APPN: ----- SS ----- Sending ACT_CP_CP_SESSION message to RCA TP
APPN: ----- SS ----- Deq ASSIGN_PCID message
APPN: ----- SS ----- Sending ASSIGN_PCID_RSP message to someone
APPN: ----- SS ----- Deq INIT_SIGNAL message
APPN: ----- SS ----- Sending REQUEST_COS_TPF_VECTOR message to TRS
APPN: ----- SS ----- Receiving an REQUEST_COS_TPF_VECTOR_RSP from TRS
APPN: ----- SS ----- Sending REQUEST_SINGLE_HOP_ROUTE message to TRS
APPN: ----- SS ----- Receiving an REQUEST_SINGLE_HOP_ROUTE_RSP from TRS
APPN: ----- SS ----- Sending ACTIVATE_ROUTE message to CS
APPN: ----- SS ----- Deq ACTIVATE_ROUTE_RSP message
APPN: ----- SS ----- Sending CINIT_SIGNAL message to SM
APPN: ----- SS ----- Deq ACT_CP_CP_SESSION_RSP message
APPN: -- SS----SS ssp00, act_cp_cp_session_rsp received, sense_code=0, cgid=5C,
ips@=A93790
APPN: Sending ADJACENT_NN_1015 to SCM, adj_node_p=A6B980,cp_name=18s
APPN: ----- SS ----- Sending ACTIVE CP_STATUS CONWINNER message to DS
APPN: ----- SS ----- Sending ACTIVE CP_STATUS CONWINNER message to MS
APPN: ----- SS ----- Sending ACTIVE CP_STATUS CONWINNER message to TRS
```

Table 2-16 describes fields in the **debug appn ss** output shown in Figure 2-24.

Table 2-16 Debug APPN SS Field Descriptions

Field	Description
APPN	APPN debugging output.
SS	SS component output.

Related Command

debug appn all

debug appn trs

Use the **debug appn trs** EXEC command to display debugging information on APPN Topology and Routing Services (TRS) component activity. The **no** form of this command disables debugging output.

[no] debug appn trs

Usage Guidelines

The Topology and Routing Services (TRS) component is responsible for creating and maintaining the topology database, creating and maintaining the class of service database, and computing and caching optimal routes through the network.

Sample Display

Figure 2-25 shows sample **debug appn trs** output.

Figure 2-25 Sample Debug APPN TRS Output

```
router# debug appn trs

Turned on event 400000FF
APPN: ----- TRS ----- Received a QUERY_CPNAME
APPN: ----- TRS ----- Received a REQUEST_ROUTE
APPN: ----- TRS ----- check_node node_name=NETA.LISA
APPN: ----- TRS ----- check_node node_index=0
APPN: ----- TRS ----- check_node node_weight=60
APPN: ----- TRS ----- add index 484 to origin description list
APPN: ----- TRS ----- add index 0 to dest description list
APPN: ----- TRS ----- origin tg_vector is NULL
APPN: ----- TRS ----- weight_to_origin = 0
APPN: ----- TRS ----- weight_to_dest = 0
APPN: ----- TRS ----- u_b_s_f weight = 30
APPN: ----- TRS ----- u_b_s_f prev_weight = 2147483647
APPN: ----- TRS ----- u_b_s_f origin_index = 484
APPN: ----- TRS ----- u_b_s_f dest_index = 0
APPN: ----- TRS ----- b_r_s_f weight = 30
APPN: ----- TRS ----- b_r_s_f origin_index = 484
APPN: ----- TRS ----- b_r_s_f dest_index = 0
APPN: ----- TRS ----- Received a REQUEST_ROUTE
APPN: ----- TRS ----- check_node node_name=NETA.LISA
APPN: ----- TRS ----- check_node node_index=0
APPN: ----- TRS ----- check_node node_weight=60
APPN: ----- TRS ----- check_node node_name=NETA.BART
APPN: ----- TRS ----- check_node node_index=484
APPN: ----- TRS ----- check_node node_weight=60
APPN: ----- TRS ----- add index 484 to origin description list
APPN: ----- TRS ----- add index 0 to dest description list
APPN: ----- TRS ----- origin_tg_weight to non-VN=30
APPN: ----- TRS ----- origin_node_weight to non-VN=60
APPN: ----- TRS ----- weight_to_origin = 90
APPN: ----- TRS ----- weight_to_dest = 0
APPN: ----- TRS ----- u_b_s_f weight = 120
APPN: ----- TRS ----- u_b_s_f prev_weight = 2147483647
APPN: ----- TRS ----- u_b_s_f origin_index = 484
APPN: ----- TRS ----- u_b_s_f dest_index = 0
APPN: ----- TRS ----- b_r_s_f weight = 120
APPN: ----- TRS ----- b_r_s_f origin_index = 484
APPN: ----- TRS ----- b_r_s_f dest_index = 0
```

Table 2-17 describes fields in the **debug appn trs** output shown in Figure 2-25.

Table 2-17 Debug APPN TRS Field Descriptions

Field	Description
APPN	APPN debugging output.
TRS	TRS component output.

Related Command

debug appn all

debug arap

Use the **debug arap** EXEC command to display AppleTalk Remote Access Protocol (ARAP) events. The **no** form of this command disables debugging output.

```
debug arap [internal]
no debug arap
```

Syntax Description

internal (Optional) Limits display to internal ARAP events.

Usage Guidelines

Use the **debug arap** command with the **debug callback** command on access servers to debug dial-in and callback events.

Sample Display

Figure 2-26 shows sample **debug arap** output.

Figure 2-26 Sample Debug ARAP Output

```
router# debug arap internal

ARAP: ----- SRVRVERSION -----
ARAP: ----- ACKing 0 -----
ARAP: ----- AUTH_CHALLENGE -----
arapsec_local_account setting up callback
ARAP: ----- ACKing 1 -----
ARAP: ----- AUTH_RESPONSE -----
arap_startup initiating callback ARAP 2.0
ARAP: ----- CALLBACK -----
TTY7 Callback process initiated, user: dialback dialstring 40
TTY7 Callback forced wait = 4 seconds
TTY7 ARAP Callback Successful - await exec/autoselect pickup
TTY7: Callback in effect
ARAP: ----- STARTINFOFROMSERVER -----
ARAP: ----- ACKing 0 -----
ARAP: ----- ZONELISTINFO -----
ARAP: ----- ZONELISTINFO -----
ARAP: ----- ZONELISTINFO -----
ARAP: ----- ZONELISTINFO -----
ARAP: ----- ZONELISTINFO -----
```

The displayed information is self-explanatory.

Related Command

debug callback

debug arp

Use the **debug arp** EXEC command to display information on Address Resolution Protocol (ARP) transactions. The **no** form of this command disables debugging output.

[no] debug arp

Usage Guidelines

Use this command when some nodes on a TCP/IP network are responding, but others are not. It shows whether the router is sending ARPs and whether it is receiving ARPs.

Sample Display

Figure 2-27 shows sample **debug arp** output.

Figure 2-27 Sample Debug ARP Output

```
router# debug arp

IP ARP: sent req src 172.16.22.7 0000.0c01.e117, dst 172.16.22.96 0000.0000.0000
IP ARP: rcvd rep src 172.16.22.96 0800.2010.b908, dst 172.16.22.7
IP ARP: rcvd req src 172.16.6.10 0000.0c00.6fa2, dst 172.16.6.62
IP ARP: rep filtered src 172.16.22.7 aa92.1b36.a456, dst 255.255.255.255 ffff.ffff.ffff
IP ARP: rep filtered src 172.16.9.7 0000.0c00.6b31, dst 172.16.22.7 0800.2010.b908
```

In Figure 2-27, each line of output represents an ARP packet that the router sent or received. Explanations for the individual lines of output follow.

The first line indicates that the router at IP address 172.16.22.7 and MAC address 0000.0c01.e117 sent an ARP request for the MAC address of the host at 172.16.22.96. The series of zeros (0000.0000.0000) following this address indicate that the router is currently unaware of the MAC address.

```
IP ARP: sent req src 172.16.22.7 0000.0c01.e117, dst 172.16.22.96 \
0000.0000.0000
```

The second line indicates that the router at IP address 172.16.22.7 receives a reply from the host at 172.16.22.96 indicating that its MAC address is 0800.2010.b908:

```
IP ARP: rcvd rep src 172.16.22.96 0800.2010.b908, dst 172.16.22.7
```

The third line indicates that the router receives an ARP request from the host at 172.16.6.10 requesting the MAC address for the host at 172.16.6.62:

```
IP ARP: rcvd req src 172.16.6.10 0000.0c00.6fa2, dst 172.16.6.62
```

The fourth line indicates that another host on the network attempted to send the router an ARP reply for its own address. The router ignores meaningless replies. Usually, meaningless replies happen if someone is running a bridge in parallel with the router and is allowing ARP to be bridged. This condition indicates a network misconfiguration.

```
IP ARP: rep filtered src 172.16.22.7 aa92.1b36.a456, dst 255.255.255.255 \
ffff.ffff.ffff
```

The fifth line indicates that another host on the network attempted to inform the router that it is on network 172.16.9.7, but the router does not know that the network is attached to a different router interface. The remote host (probably a PC or an X terminal) is misconfigured. If the router were to install this entry, it would deny service to the real machine on the proper cable. |

```
IP ARP: rep filtered src 172.16.9.7 0000.0c00.6b31, dst 172.16.22.7 \
0800.2010.b908 |
```

debug atm errors

Use the **debug atm errors** EXEC command to display Asynchronous Transfer Mode (ATM) errors. The **no** form of this command disables debugging output.

[no] debug atm errors

Sample Display

Figure 2-28 shows sample **debug atm errors** output.

Figure 2-28 Sample Debug ATM Errors Output

```
router# debug atm errors  
ATM(ATM2/0): Encapsulation error, link=7, host=836CA86D.  
ATM(ATM4/0): VCD#7 failed to echo OAM. 4 tries
```

The first line of output in Figure 2-28 indicates that a packet was routed to the ATM interface, but no static map was set up to route that packet to the proper virtual circuit.

The second line of output shows that an OAM F5 (virtual circuit) cell error occurred.

debug atm events

Use the **debug atm events** EXEC command to display ATM events. The **no** form of this command disables debugging output.

[no] debug atm events

Usage Guidelines

This command displays ATM events that occur on the ATM interface processor and is useful for diagnosing problems in an ATM network. It provides an overall picture of the stability of the network. In a stable network, the **debug atm events** command does not return any information. If the command generates numerous messages, the messages can indicate the possible source of problems.

When configuring or making changes to a router or interface for ATM, enable **debug atm events**. Doing so alerts you to the progress of the changes or to any errors that might result. Also use this command periodically when you suspect network problems.

Sample Display

Figure 2-29 shows sample **debug atm events** output.

Figure 2-29 Sample Debug ATM Events Output

```
router# debug atm events
ATM events debugging is on
RESET(ATM4/0): PLIM type is 1, Rate is 100Mbps
aip_disable(ATM4/0): state=1
config(ATM4/0)
aip_love_note(ATM4/0): asr=0x201
aip_enable(ATM4/0)
aip_love_note(ATM4/0): asr=0x4000
aip_enable(ATM4/0): restarting VCs: 7
aip_setup_vc(ATM4/0): vc:1 vpi:1 vci:1
aip_love_note(ATM4/0): asr=0x200
aip_setup_vc(ATM4/0): vc:2 vpi:2 vci:2
aip_love_note(ATM4/0): asr=0x200
aip_setup_vc(ATM4/0): vc:3 vpi:3 vci:3
aip_love_note(ATM4/0): asr=0x200
aip_setup_vc(ATM4/0): vc:4 vpi:4 vci:4
aip_love_note(ATM4/0): asr=0x200
aip_setup_vc(ATM4/0): vc:6 vpi:6 vci:6
aip_love_note(ATM4/0): asr=0x200
aip_setup_vc(ATM4/0): vc:7 vpi:7 vci:7
aip_love_note(ATM4/0): asr=0x200
aip_setup_vc(ATM4/0): vc:11 vpi:11 vci:11
aip_love_note(ATM4/0): asr=0x200
```

Table 2-18 describes significant fields in the output shown in Figure 2-29.

Table 2-18 Debug ATM Events Field Descriptions

Field	Description
PLIM type	Indicates the interface rate in Mbps. Possible values are <ul style="list-style-type: none"> • 1 = TAXI(4B5B) 100 Mbps • 2 = SONET 155 Mbps • 3 = E3 34 Mbps
state	Indicates current state of the AIP. Possible values are <ul style="list-style-type: none"> • 1 = An ENABLE will be issued soon • 0 = The AIP will remain shut down
asr	Defines a bitmask, which indicates actions or completions to commands. Valid bitmask values are <ul style="list-style-type: none"> • 0x0800 = AIP crashed, reload may be required. • 0x0400 = AIP detected a carrier state change. • 0x0n00 = Command completion status. Command completion status codes are <ul style="list-style-type: none"> — n = 8 Invalid PLIM detected — n = 4 Command failed — n = 2 Command completed successfully — n = 1 CONFIG request failed — n = 0 Invalid value

Explanations for representative lines of output in Figure 2-29 follow.

The following line indicates that the ATM Interface Processor (AIP) was reset. The PLIM TYPE detected was 1, so the maximum rate is set to 100 Mbps.

```
RESET(ATM4/0): PLIM type is 1, Rate is 100Mbps
```

The following line indicates that the ATM Interface Processor (AIP) was given a **shutdown** command, but the current configuration indicates that the AIP should be up:

```
aip_disable(ATM4/0): state=1
```

The following line indicates that a configuration command has been completed by the AIP:

```
aip_love_note(ATM4/0): asr=0x201
```

The following line indicates that the AIP was given a **no shutdown** command to take it out of shutdown:

```
aip_enable(ATM4/0)
```

The following line indicates that the AIP detected a carrier state change. It does not indicate that the carrier is down or up, only that it has changed:

```
aip_love_note(ATM4/0): asr=0x4000
```

The following line of output indicates that the AIP enable function is restarting all PVCs automatically:

```
aip_enable(ATM4/0): restarting VCs: 7
```

The following lines of output indicate that PVC 1 was set up and a successful completion code was returned:

```
aip_setup_vc(ATM4/0): vc:1 vpi:1 vci:1  
aip_love_note(ATM4/0): asr=0x200
```

debug atm oam

Use the **debug atm oam** EXEC command to display ATM operation and maintenance (OAM) events. The **no** form of this command disables debugging output.

[no] debug atm oam

Sample Display

Figure 2-30 shows sample **debug atm oam** output.

Figure 2-30 Sample Debug ATM OAM Output

```
router# debug atm oam

ATM4/0(O): VCD:0x0 DM:0x300 *OAM Cell* Length:0x39
0000 0300 0070 007A 0018 0100 0000 05FF FFFF FFFF FFFF FFFF FFFF FFFF FFFF
FFFF FFFF FFFF FFFF FF6A 6A6A 6A6A 6A6A 6A6A 6A6A 6A6A 6A6A 6A00 0000
```

Table 2-19 describes the output fields shown in Figure 2-30.

Table 2-19 Debug ATM OAM Field Descriptions

Field	Description
0000	VCD Special OAM indicator
0300	Descriptor MODE bits for the AIP
0	GFC (4 bits)
07	VPI (8 bits)
0007	VCI (16 bits)
A	Payload type field(PTI)(4 bits)
00	Header Error Correction(8 bits)
1	OAM Fault mgmt. cell(4 bits)
8	OAM LOOPBACK indicator (4 bits)
01	Loopback indicator value, always 1(8 bits)
00000005	Loopback unique ID, sequence number (32 bits)
FF6A	F's and 6A required in the remaining ATM cell, per UNI3.0

debug atm packet

Use the **debug atm packet** EXEC command to display per-packet debugging output. The output reports information online when a packet is received or a transmit is attempted. The **no** form of this command disables debugging output.

```
[no] debug atm packet [interface atm number [vcd vcd-number]]
```

Syntax Description

interface *number* (Optional) ATM interface or subinterface number.

vcd *vcd-number* (Optional) Number of the virtual circuit designator (VCD).

Usage Guidelines

The **debug atm packet** command displays all process-level ATM packets for both outbound and inbound packets. This command is useful for determining whether packets are being received and transmitted correctly.

For transmitted packets, the information is displayed only after the protocol data unit (PDU) is entirely encapsulated and a next hop virtual circuit (VC) is found. If information is not displayed, the address translation probably failed during encapsulation. When a next hop VC is found, the packet is displayed exactly as it will be presented on the wire. Having a display indicates the packets are properly encapsulated for transmission.

For received packets, information is displayed for all incoming frames. The display can show whether the transmitting station properly encapsulates the frames. Because all incoming frames are displayed, this information is useful when performing back-to-back testing and corrupted frames cannot be dropped by an intermediary ATM switch.

The **debug atm packet** command also displays the initial bytes of the actual PDU in hexadecimal. This information can be decoded only by qualified support or engineering personnel.

Note Because the **debug atm packet** command generates a significant amount of output for every packet processed, use it only when traffic on the network is low, so other activity on the system is not adversely affected.

Sample Display

Figure 2-31 shows sample **debug atm packet** output.

Figure 2-31 Sample Debug ATM Packet Output

```
router# debug atm packet
ATM packets debugging is on
router#
ATM2/0(O): VCD: 0x1,DM: 1C00, MUX, ETYPE: 0800,Length: 32
4500 002E 0000 0000 0209 92ED 836C A26E FFFF FFFF 1108 006D 0001 0000 0000
A5CC 6CA2 0000 000A 0000 6411 76FF 0100 6C08 00FF FFFF 0003 E805 DCFF 0105
```

Table 2-20 describes significant fields shown in Figure 2-31.

Table 2-20 **Debug ATM Packet Field Descriptions**

Field	Description
ATM2/0	Indicates the interface that generated this packet.
(O)	Indicates an output packet. (I) would mean receive packet.
VCD: 0xn	Indicates the virtual circuit associated with this packet, where <i>n</i> is some value.
DM: 0xnnnn	Indicates the descriptor mode bits on output only, where <i>nnnn</i> is a hexadecimal value.
ETYPE: <i>n</i>	Shows the Ethernet type for this packet.
Length: <i>n</i>	Shows the total length of the packet including the ATM header(s).

The following two lines of output are the binary data, which are the contents of the protocol PDU before encapsulation at the ATM:

```
4500 002E 0000 0000 0209 92ED 836C A26E FFFF FFFF 1108 006D 0001 0000 0000
A5CC 6CA2 0000 000A 0000 6411 76FF 0100 6C08 00FF FFFF 0003 E805 DCFE 0105
```

debug bri

Use the **debug bri** EXEC command to display debugging information on Integrated Services Digital Networks (ISDN) Basic Rate Interface (BRI) routing activity. The **no** form of this command disables debugging output.

[no] debug bri

Usage Guidelines

The **debug bri** command indicates whether the ISDN code is enabling and disabling the B-channels when attempting an outgoing call. This command is available for the low-end router products that have a multi-BRI network interface module installed.

Note Because the **debug bri** command generates a significant amount of output, use it only when traffic on the IP network is low, so other activity on the system is not adversely affected.

Sample Display

Figure 2-32 shows sample **debug bri** output.

Figure 2-32 Sample Debug BRI Packets Output

```
Router# debug bri

Basic Rate network interface debugging is on
BRI: write_sid: wrote 1B for subunit 0, slot 1.
BRI: write_sid: wrote 15 for subunit 0, slot 1.
BRI: write_sid: wrote 17 for subunit 0, slot 1.
BRI: write_sid: wrote 6 for subunit 0, slot 1.
BRI: write_sid: wrote 8 for subunit 0, slot 1.
BRI: write_sid: wrote 11 for subunit 0, slot 1.
BRI: write_sid: wrote 13 for subunit 0, slot 1.
BRI: write_sid: wrote 29 for subunit 0, slot 1.
BRI: write_sid: wrote 1B for subunit 0, slot 1.
BRI: write_sid: wrote 15 for subunit 0, slot 1.
BRI: write_sid: wrote 17 for subunit 0, slot 1.
BRI: write_sid: wrote 20 for subunit 0, slot 1.
BRI: Starting Power Up timer for unit = 0.
BRI: write_sid: wrote 3 for subunit 0, slot 1.
BRI: Starting T3 timer after expiry of PUP timeout for unit = 0, current state is F4.
BRI: write_sid: wrote FF for subunit 0, slot 1.
BRI: Activation for unit = 0, current state is F7.
BRI: enable channel B1
BRI: write_sid: wrote 14 for subunit 0, slot 1.

%LINK-3-UPDOWN: Interface BRI0: B-Channel 1, changed state to up
%LINK-5-CHANGED: Interface BRI0: B-Channel 1, changed state to up.!!!
BRI: disable channel B1
BRI: write_sid: wrote 15 for subunit 0, slot 1.

%LINK-3-UPDOWN: Interface BRI0: B-Channel 1, changed state to down
%LINK-5-CHANGED: Interface BRI0: B-Channel 1, changed state to down
%LINEPROTO-5-UPDOWN: Line protocol on Interface BRI0: B-Channel 1, changed state to down
```

Explanations for individual lines of output from Figure 2-32 follow.

The following line indicates that an internal command was written to the interface controller. The subunit identifies the first interface in the slot:

```
BRI: write_sid: wrote 1B for subunit 0, slot 1.
```

The following line indicates that the power-up timer was started for the named unit:

```
BRI: Starting Power Up timer for unit = 0.
```

The following lines indicate that the channel or the protocol on the interface changed state:

```
%LINK-3-UPDOWN: Interface BRI0: B-Channel 1, changed state to up
%LINK-5-CHANGED: Interface BRI0: B-Channel 1, changed state to up.!!!
%LINEPROTO-5-UPDOWN: Line protocol on Interface BRI0: B-Channel 1, changed state to down
```

The following line indicates that the channel was disabled:

```
BRI: disable channel B1
```

Lines of output not described are for use by support staff only.

Related Commands

debug isdn event
debug isdn q921
debug isdn q931

debug bsc event

Use the **debug bsc event EXEC** command to display all events occurring in the Binary Synchronous Communications (BSC) feature. The **no** form of this command disables debugging output.

[no] debug bsc event [*number*]

Syntax Description

number (Optional) Group number.

Usage Guidelines

This command traces all interfaces configured with a **bsc protocol-group** *number* command.

Sample Display

Figure 2-33 shows sample **debug bsc event** output.

Figure 2-33 Sample Debug BSC Event Output

```
router# debug bsc event

BSC: Serial2          POLLEE-FSM inp:E_LineFail old_st:CU_Down new_st:TCU_EOFFile
BSC: Serial2          POLLEE-FSM inp:E_LineFail old_st:CU_Down new_st:TCU_EOFFile
BSC: Serial2          POLLEE-FSM inp:E_LineFail old_st:CU_Down new_st:TCU_EOFFile
0:04:32: BSC: Serial2 :SDI-rx: 9 bytes
BSC: Serial2          POLLEE-FSM inp:E_RxEtx old_st:CU_Down new_st:TCU_EOFFile
0:04:32: BSC: Serial2 :SDI-rx: 5 bytes
BSC: Serial2          POLLEE-FSM inp:E_RxEng old_st:CU_Down new_st:TCU_EOFFile
BSC: Serial2          POLLEE-FSM inp:E_Timeout old_st:CU_Down new_st:TCU_InFile
BSC: Serial2          POLLEE-FSM inp:E_Timeout old_st:CU_Idle new_st:TCU_InFile
%LINEPROTO-5-UPDOWN: Line protocol on Interface Serial2, changed state to up
%LINK-3-UPDOWN: Interface Serial2, changed state to up
BSC: Serial2          POLLEE-FSM inp:E_Timeout old_st:CU_Idle new_st:TCU_InFile
0:04:35: BSC: Serial2 :SDI-rx: 9 bytes
BSC: Serial2          POLLEE-FSM inp:E_RxEtx old_st:CU_Idle new_st:TCU_InFile
0:04:35: BSC: Serial2 :SDI-rx: 5 bytes
BSC: Serial2          POLLEE-FSM inp:E_RxEng old_st:CU_Idle new_st:TCU_InFile
0:04:35: BSC: Serial2 :NDI-rx: 3 bytes
```

Related Commands

debug bsc packet
debug bstun events

debug bsc packet

Use the **debug bsc packet** EXEC command to display all frames traveling through the Binary Synchronous Communications (BSC) feature. The **no** form of this command disables debugging output.

[no] debug bsc packet [group number] [buffer-size bytes]

Syntax Description

group number	(Optional) Group number.
buffer-size bytes	(Optional) Number of bytes displayed per packet (defaults to 20).

Usage Guidelines

This command traces all interfaces configured with a **bsc protocol-group number** command.

Sample Display

Figure 2-34 shows sample **debug bsc packet** output.

Figure 2-34 Sample Debug BSC Packet Output

```
router# debug bsc packet
0:23:33: BSC: Serial2      :NDI-rx : 27 bytes 401A400227F5C31140C11D60C8C5D3D3D51D4013
0:23:33: BSC: Serial2      :SDI-tx  : 12 bytes 00323237FF3232606040402D
0:23:33: BSC: Serial2      :SDI-rx  : 2 bytes 1070
0:23:33: BSC: Serial2      :SDI-tx  : 27 bytes 401A400227F5C31140C11D60C8C5D3D3D51D4013
0:23:33: BSC: Serial2      :SDI-rx  : 2 bytes 1061
0:23:33: BSC: Serial2      :SDI-tx  : 5 bytes 00323237FF
```

Related Commands

debug bsc event
debug bstun events

debug bstun events

Use the **debug bstun events** EXEC command to display BSTUN connection events and status. The **no** form of this command disables debugging output.

[no] debug bstun events *number*

Syntax Description

number (Optional) Group number.

Usage Guidelines

When you enable the **debug bstun events** command, messages showing connection establishment and other overall status messages are displayed.

You can use the **debug bstun events** command to assist you in determining whether the BSTUN peers are configured correctly and are communicating. For example, if you enable the **debug bstun packet** command and you do not see any packets, you may want to enable event debugging.

Note Also refer to the **debug bsc packet** and **debug bsc event** commands. Currently, these two commands support the only protocol working through the BSTUN tunnel. Sometimes frames do not go through the tunnel because they have been discarded at the BSC protocol level.

Sample Displays

Figure 2-35 shows a sample **debug bstun events** output of keepalive messages working correctly. If the routers are configured correctly, at least one router will show reply messages.

Figure 2-35 Sample Debug BSTUN Events Output—Keepalive Messages

```
BSTUN: Received Version Reply opcode from (all[2])_172.16.12.2/1976 at 1360
BSTUN: Received Version Request opcode from (all[2])_172.16.12.2/1976 at 1379
BSTUN: Received Version Reply opcode from (all[2])_172.16.12.2/1976 at 1390
```

Note In a scenario where there is constantly loaded bi-directional traffic, you might not see keepalive messages because they are sent only when the remote end has been silent for the keepalive period.

Figure 2-36 shows a sample **debug bstun events** output of an event trace in which the wrong TCP address has been specified for the remote peer. These are non-keepalive related messages.

Figure 2-36 Sample Debug BSTUN Events Output—Event Trace

```
BSTUN: Change state for peer (C1[1])172.16.12.22/1976 (closed->opening)
BSTUN: Change state for peer (C1[1])172.16.12.22/1976 (opening->open wait)
%BSTUN-6-OPENING: CONN: opening peer (C1[1])172.16.12.22/1976, 3
BSTUN: tcpd sender in wrong state, dropping packet
BSTUN: tcpd sender in wrong state, dropping packet
```

BSTUN: tcpd sender in wrong state, dropping packet

Related Commands

debug bsc event

debug bsc packet

debug bstun packet

debug bstun packet

Use the **debug bstun packet** EXEC command to display packet information on packets traveling through the BSTUN links. The **no** form of this command disables debugging output.

[no] debug bstun packet [group number] [buffer-size bytes]

Syntax Description

group number	(Optional) BSTUN group number.
buffer-size bytes	(Optional) Number of bytes displayed per packet (defaults to 20).

Sample Display

Figure 2-37 shows sample **debug bstun packet** output.

Figure 2-37 Sample Debug BSTUN Packet Output

```
router# debug bstun packet
BSTUN bsc-local-ack: 0:00:00 Serial2      SDI: Addr: 40 Data: 02C1C1C1C1C1C1C1C1
BSTUN bsc-local-ack: 0:00:00 Serial2      SDI: Addr: 40 Data: 02C1C1C1C1C1C1C1C1
BSTUN bsc-local-ack: 0:00:06 Serial2      NDI: Addr: 40 Data: 0227F5C31140C11D60C8
```

Related Command

debug bstun events

debug callback

Use the **debug callback EXEC** command to display callback events when the router is using a modem and a chat script to call back on a terminal line. The **no** form of this command disables debugging output.

[no] debug callback

Usage Guidelines

This command is useful for debugging chat scripts on PPP and ARAP lines that use callback mechanisms. The output provided by the **debug callback** command shows you how the call is progressing when used with the **debug ppp** or **debug arap** commands.

Sample Display

Figure 2-38 shows sample **debug callback** output.

Figure 2-38 Sample Debug Callback Output

```
router# debug callback
TTY7 Callback process initiated, user: exec_test dialstring 123456
TTY7 Callback forced wait = 4 seconds
TTY7 Exec Callback Successful - await exec/autoselect pickup
TTY7: Callback in effect
```

Related Commands

debug arap

debug ppp

debug cdp

Use the **debug cdp** EXEC command to enable debugging of Cisco Discovery Protocol (CDP). The **no** form of this command disables debugging output.

```
[no] debug cdp {packets | adjacency | events}
```

Syntax Description

packets	Enables packet-related debugging output.
adjacency	Enables adjacency-related debugging output.
events	Enables output related to error messages, such as detecting a bad checksum.

Usage Guidelines

Use **debug cdp** commands to display information about CDP packet activity, activity between CDP neighbors, and various CDP events.

Sample Display

Figure 2-39 shows a composite sample output from **debug cdp packets**, **debug cdp adjacency**, and **debug cdp events**.

Figure 2-39 Sample Debug CDP Output

```
router# debug cdp packets
CDP packet info debugging is on
router# debug cdp adjacency
CDP neighbor info debugging is on
router# debug cdp events
CDP events debugging is on

CDP-PA: Packet sent out on Ethernet0
CDP-PA: Packet received from gray.cisco.com on interface Ethernet0

CDP-AD: Deleted table entry for violet.cisco.com, interface Ethernet0
CDP-AD: Interface Ethernet2 coming up

CDP-EV: Encapsulation on interface Serial2 failed
```

The messages displayed by **debug cdp** commands are self-explanatory.

debug cdp ip

Use the **debug cdp ip** EXEC command to enable debug output for the IP routing information that is carried and processed by the Cisco Discovery Protocol (CDP). The **no** form of this command disables debugging output.

[no] debug cdp ip

Usage Guidelines

CDP is a media- and protocol-independent device-discovery protocol that runs on all Cisco routers.

You can use the **debug cdp ip** command to determine the IP network prefixes CDP is advertising and whether CDP is correctly receiving this information from neighboring routers.

Use the **debug cdp ip** command with the **debug ip routing** command to debug problems that occur when on-demand routing (ODR) routes are not installed in the routing table at a hub router. You can also use the **debug cdp ip** command with the **debug cdp packet** and **debug cdp adjacency** commands along with encapsulation-specific debug commands to debug problems that occur in the receipt of CDP IP information.

Sample Display

Figure 2-40 shows sample **debug cdp ip** output. This example shows the transmission of IP-specific information in a CDP update. In this case, three network prefixes are being transmitted, each with a different network mask.

Figure 2-40 Sample Debug CDP IP Output

```
router# debug cdp ip
CDP-IP: Writing prefix 172.1.69.232.112/28
CDP-IP: Writing prefix 172.19.89.0/24
CDP-IP: Writing prefix 11.0.0.0/8
```

In addition to the messages shown in Figure 2-40, you might see the following messages:

- This message indicates that CDP is attempting to install the prefix 172.1.1.0/24 into the IP routing table:
CDP-IP: Updating prefix 172.1.1.0/24 in routing table
- This message indicates a protocol error occurred during an attempt to decode an incoming CDP packet:
CDP-IP: IP TLV length (3) invalid
- This message indicates the receipt of the IP prefix 172.1.1.0/24 from a CDP neighbor connected via the Ethernet interface 0/0. The neighbor's IP address is 10.0.0.1.

```
CDP-IP: Reading prefix 172.1.1.0/24 source 10.0.0.1 via Ethernet0/0
```

Related Commands

debug cdp adjacency

debug cdp packet

debug ip routing

debug channel events

The **debug channel events** EXEC command displays processing events that occur on the channel adapter interfaces of all installed adapters. This command is valid for the Cisco 7000 series routers only. The **no** form of this command disables debugging output.

[no] debug channel events

Usage Guidelines

This command displays Channel Interface Processor (CIP) events that occur on the CIP interface processor and is useful for diagnosing problems in an IBM channel attach network. It provides an overall picture of the stability of the network. In a stable network, the **debug channel events** command does not return any information. If the command generates numerous messages, they can indicate the possible source of the problems. To observe the statistic message (cip_love_letter) transmitted every ten seconds, use the **debug channel love** command.

When configuring or making changes to a router or interface that supports IBM channel attach, enable **debug channel events**. Doing so alerts you to the progress of the changes or to any errors that might result. Also use this command periodically when you suspect network problems.

Sample Display

Figure 2-41 shows sample **debug channel events** output.

Figure 2-41 Sample Debug Channel Events Output

```
Router# debug channel events

Channel3/0: cip_reset(), state administratively down
Channel3/0: cip_reset(), state up
Channel3/0: sending nodeid
Channel3/0: sending command for vc 0, CLAW path C700, device C0
```

Explanations for individual lines of output from Figure 2-41 follow.

The following line indicates that the CIP is being reset to an administrative down state:

```
Channel3/0: cip_reset(), state administratively down
```

The following line indicates that the CIP is being reset to an administrative up state:

```
Channel3/0: cip_reset(), state up
```

The following line indicates that the node id is being sent to the CIP. This information is the same as the “Local Node” information under the **show extended channel slot/port subchannels** command. The CIP needs this information to send to the host mainframe.

```
Channel3/0: sending nodeid
```

The following line indicates that a CLAW subchannel command is being sent from the RP to the CIP. The value vc 0 indicates that the CIP will use virtual circuit number 0 with this device. The virtual circuit number will also show up when you use the **debug channel packets** command.

```
Channel3/0: sending command for vc 0, CLAW path C700, device C0
```

Related Commands

debug channel love

debug channel packets

debug channel love

Use the **debug channel love** EXEC command to display Channel Interface Processor (CIP) love letter events. This command is valid for the Cisco 7000 series routers only. The **no** form of this command disables debugging output.

[no] debug channel love

Usage Guidelines

This command displays Channel Interface Processor (CIP) events that occur on the CIP interface processor and is useful for diagnosing problems in an IBM channel attach network. It provides an overall picture of the stability of the network. In a stable network, the **debug channel love** command returns a statistic message (`cip_love_letter`) that is transmitted every ten seconds.

Sample Display

Figure 2-42 shows sample **debug channel love** output.

Figure 2-42 Sample Debug Channel Love Output

```
Router# debug channel love
Channel3/1: love letter received, bytes 3308
Channel3/0: love letter received, bytes 3336
cip_love_letter: received 11, but no cip_info
```

The following line indicates that data was received on the CIP:

```
Channel3/1: love letter received, bytes 3308
```

The following line indicates that the interface is enabled, but there is no configuration for it. It does not normally indicate a problem, just that the route processor (RP) got statistics from the CIP but has no place to store them.

```
cip_love_letter: recieved 11, but no cip_info
```

Related Commands

debug channel events

debug channel packets

debug channel packets

Use the **debug channel packets** EXEC command to display per-packet debugging output. The output reports information when a packet is received or a transmit is attempted. The **no** form of this command disables debugging output.

[no] debug channel packets

Usage Guidelines

The **debug channel packets** command displays all process-level Channel Interface Processor (CIP) packets for both outbound and inbound packets. You will need to disable fast switching and autonomous switching to obtain debugging output. This command is useful for determining whether packets are received or transmitted correctly.

This command is valid for the Cisco 7000 series routers only.

Sample Display

Figure 2-43 shows sample **debug channel packets** output.

Figure 2-43 Sample Debug Channel Packets Output

```
Router# debug channel packets

Channel packets debugging is on
(Channel3/0)-out size = 104, vc = 0000, type = 0800, src 172.24.0.11, dst 172.24.1.58
(Channel3/0)-in size = 48, vc = 0000, type = 0800, src 172.24.1.58, dst 172.24.15.197
(Channel3/0)-in size = 48, vc = 0000, type = 0800, src 172.24.1.58, dst 172.24.15.197
(Channel3/0)-out size = 71, vc = 0000, type = 0800, src 172.24.15.197, dst 172.24.1.58
(Channel3/0)-in size = 44, vc = 0000, type = 0800, src 172.24.1.58, dst 172.24.15.197
```

Table 2-21 provides explanations for individual lines of output from Figure 2-43.

Table 2-21 Channel Packets Field Descriptions

Field	Description
(Channel3/0)	The interface slot and port.
in / out	In is a packet from the mainframe to the router. Out is a packet from the router to the mainframe.
size =	The number of bytes in the packet, including internal overhead.
vc =	A value from 0–511 that maps to the claw interface configuration command. This information is from the MAC layer.
type =	The encapsulation type in the MAC layer. The value 0800 indicates an IP datagram.
src	The origin, or source, of the packet, as opposed to the previous hop address.
dst	The destination of the packet, as opposed to the next hop address.

Related Commands

debug channel events

debug channel love

debug clns esis events

Use the **debug clns esis events** EXEC command to display uncommon End System-to-Intermediate System (ES-IS) events, including previously unknown neighbors, neighbors that have aged out, and neighbors that have changed roles (ES to IS, for example). The **no** form of this command disables debugging output.

[no] debug clns esis events

Sample Display

Figure 2-44 shows sample **debug clns esis events** output.

Figure 2-44 Sample Debug CLNS ESIS Events Output

```
router# debug clns esis events

ES-IS: ISH from aa00.0400.2c05 (Ethernet1), HT 30
ES-IS: ESH from aa00.0400.9105 (Ethernet1), HT 150
ES-IS: ISH sent to All ESs (Ethernet1): NET 49.0001.AA00.0400.6904.00, HT 299, HLEN 20
```

Explanations for individual lines of output from Figure 2-44 follow.

The following line indicates that the router received a hello packet (ISH) from the IS at MAC address aa00.0400.2c05 on the Ethernet1 interface. The hold time (or number of seconds to consider this packet valid before deleting it) for this packet is 30 seconds.

```
ES-IS: ISH from aa00.0400.2c05 (Ethernet1), HT 30
```

The following line indicates that the router received a hello packet (ESH) from the ES at MAC address aa00.0400.9105 on the Ethernet1 interface. The hold time is 150 seconds.

```
ES-IS: ESH from aa00.0400.9105 (Ethernet1), HT 150
```

The following line indicates that the router sent an IS hello packet on the Ethernet0 interface to all ESs on the network. The network entity title (NET) address of the router is 49.0001.0400.AA00.6904.00; the hold time for this packet is 299 seconds; and the header length of this packet is 20 bytes.

```
ES-IS: ISH sent to All ESs (Ethernet1): NET 49.0001.AA00.0400.6904.00, HT 299, HLEN 20
```

debug clns esis packets

Use the **debug clns esis packets** EXEC command to enable display information on End System-to-Intermediate System (ES-IS) packets that the router has received and sent. The **no** form of this command disables debugging output.

[no] debug clns esis packets

Sample Display

Figure 2-45 shows sample **debug clns esis packets** output.

Figure 2-45 Sample Debug CLNS ESIS Packets Output

```
router# debug clns esis packets

ES-IS: ISH sent to All ESs (Ethernet0): NET
47.0005.80ff.ef00.0000.0001.5940.1600.8906.4023.00, HT 299, HLEN 33
ES-IS: ISH sent to All ESs (Ethernet1): NET
47.0005.80ff.ef00.0000.0001.5940.1600.8906.4023.00, HT 299, HLEN 34
ES-IS: ISH from aa00.0400.6408 (Ethernet0), HT 299
ES-IS: ISH sent to All ESs (Tunnel0): NET
47.0005.80ff.ef00.0000.0001.5940.1600.0906.4023.00, HT 299, HLEN 34
IS-IS: ESH from 0000.0c00.bda8 (Ethernet0), HT 300
```

Explanations for individual lines of output from Figure 2-45 follow.

The following line indicates that the router has sent an IS hello packet on Ethernet0 to all ESs on the network. This hello packet indicates that the NET of the router is 47.0005.80ff.ef00.0000.0001.5940.1600.8906.4023.00. The hold time for this packet is 299 seconds. The packet header is 33 bytes in length.

```
ES-IS: ISH sent to All ESs (Ethernet0): NET
47.0005.80ff.ef00.0000.0001.5940.1600.8906.4023.00, HT 299, HLEN 33
```

The following line indicates that the router has sent an IS hello packet on Ethernet1 to all ESs on the network. This hello packet indicates that the network entity title (NET) of the router is 47.0005.80ff.ef00.0000.0001.5940.1600.8906.4023.00. The hold time for this packet is 299 seconds. The packet header is 34 bytes in length.

```
ES-IS: ISH sent to All ESs (Ethernet1): NET
47.0005.80ff.ef00.0000.0001.5940.1600.8906.4023.00, HT 299, HLEN 34
```

The following line indicates that the router received a hello packet on Ethernet0 from an intermediate system, aa00.0400.6408. The hold time for this packet is 299 seconds.

```
ES-IS: ISH from aa00.0400.6408 (Ethernet0), HT 299
```

The following line indicates that the router has sent an IS hello packet on Tunnel0 to all ESs on the network. This hello packet indicates that the NET of the router is 47.0005.80ff.ef00.0000.0001.5940.1600.8906.4023.00. The hold time for this packet is 299 seconds. The packet header is 34 bytes in length.

```
ES-IS: ISH sent to All ESs (Tunnel0): NET
47.0005.80ff.ef00.0000.0001.5940.1600.8906.4023.00, HT 299, HLEN 34
```

The following line indicates that on Ethernet0, the router received a hello packet from an end system with an SNPA of 0000.0c00.bda8. The hold time for this packet is 300 seconds.

```
IS-IS: ESH from 0000.0c00.bda8 (Ethernet0), HT 300
```

debug clns events

Use the **debug clns events** EXEC command to display CLNS events that are occurring at the router. The **no** form of this command disables debugging output.

[no] debug clns events

Sample Display

Figure 2-46 shows sample **debug clns events** output.

Figure 2-46 Sample Debug CLNS Events Output

```
router# debug clns events

CLNS: Echo PDU received on Ethernet3 from 39.0001.2222.2222.2222.00!
CLNS: Sending from 39.0001.3333.3333.3333.00 to 39.0001.2222.2222.2222.00
      via 2222.2222.2222 (Ethernet3 0000.0c00.3a18)
CLNS: Forwarding packet size 117
      from 39.0001.2222.2222.2222.00
      to 49.0002.0001.AAAA.AAAA.AAAA.00
      via 49.0002 (Ethernet3 0000.0c00.b5a3)
CLNS: RD Sent on Ethernet3 to 39.0001.2222.2222.2222.00 @ 0000.0c00.3a18,
      redirecting 49.0002.0001.AAAA.AAAA.AAAA.00 to 0000.0c00.b5a3
```

Explanations for individual lines of output from Figure 2-46 follow.

The following line indicates that the router received an echo PDU on Ethernet3 from source network service access point (NSAP) 39.0001.2222.2222.2222.00. The exclamation point at the end of the line has no significance.

```
CLNS: Echo PDU received on Ethernet3 from 39.0001.2222.2222.2222.00!
```

The following lines indicate that the router at source NSAP 39.0001.3333.3333.3333.00 is sending a CLNS echo packet to destination NSAP 39.0001.2222.2222.2222.00 via an IS with system ID 2222.2222.2222. The packet is being sent on the Ethernet3 interface, with a MAC address of 0000.0c00.3a18.

```
CLNS: Sending from 39.0001.3333.3333.3333.00 to 39.0001.2222.2222.2222.00
      via 2222.2222.2222 (Ethernet3 0000.0c00.3a18)
```

The following lines indicate that a CLNS echo packet 117 bytes in size is being sent from source NSAP 39.0001.2222.2222.2222.00 to destination NSAP 49.0002.0001.AAAA.AAAA.AAAA.00 via the router at NSAP 49.0002. The packet is being forwarded on the Ethernet3 interface, with a MAC address of 0000.0c00.b5a3.

```
CLNS: Forwarding packet size 117
      from 39.0001.2222.2222.2222.00
      to 49.0002.0001.AAAA.AAAA.AAAA.00
      via 49.0002 (Ethernet3 0000.0c00.b5a3)
```

The following lines indicate that the router sent a redirect packet on the Ethernet3 interface to the NSAP 39.0001.2222.2222.2222.00 at MAC address 0000.0c00.3a18 to indicate that NSAP 49.0002.0001.AAAA.AAAA.AAAA.00 can be reached at MAC address 0000.0c00.b5a3.

```
CLNS: RD Sent on Ethernet3 to 39.0001.2222.2222.2222.00 @ 0000.0c00.3a18,
      redirecting 49.0002.0001.AAAA.AAAA.AAAA.00 to 0000.0c00.b5a3
```

debug clns igrp packets

Use the **debug clns igrp packets** EXEC command to display debugging information on all ISO-IGRP routing activity. The **no** form of this command disables debugging output.

[no] debug clns igrp packets

Sample Display

Figure 2-47 shows sample **debug clns igrp packets** output.

Figure 2-47 Sample Debug CLNS IGRP Packets Output

```
router# debug clns igrp packets

ISO-IGRP: Hello sent on Ethernet3 for DOMAIN_green1
ISO-IGRP: Received hello from 39.0001.3333.3333.3333.00, (Ethernet3), ht 51
ISO-IGRP: Originating level 1 periodic update
ISO-IGRP: Advertise dest: 2222.2222.2222
ISO-IGRP: Sending update on interface: Ethernet3
ISO-IGRP: Originating level 2 periodic update
ISO-IGRP: Advertise dest: 0001
ISO-IGRP: Sending update on interface: Ethernet3
ISO-IGRP: Received update from 3333.3333.3333 (Ethernet3)
ISO-IGRP: Opcode: area
ISO-IGRP: Received level 2 adv for 0001 metric 1100
ISO-IGRP: Opcode: station
ISO-IGRP: Received level 1 adv for 3333.3333.3333 metric 1100
```

Explanations for individual lines of output from Figure 2-47 follow.

The following line indicates that the router is sending a hello packet to advertise its existence in the DOMAIN_green1 domain:

```
ISO-IGRP: Hello sent on Ethernet3 for DOMAIN_green1
```

The following line indicates that the router received a hello packet from a certain network service access point (NSAP) on the Ethernet3 interface. The hold time for this information is 51 seconds.

```
ISO-IGRP: Received hello from 39.0001.3333.3333.3333.00, (Ethernet3), ht 51
```

The following lines indicate that the router is generating a Level 1 update to advertise reachability to destination NSAP 2222.2222.2222 and that it is sending that update to all systems that can be reached through the Ethernet3 interface:

```
ISO-IGRP: Originating level 1 periodic update
ISO-IGRP: Advertise dest: 2222.2222.2222
ISO-IGRP: Sending update on interface: Ethernet3
```

The following lines indicate that the router is generating a Level 2 update to advertise reachability to destination area 1 and that it is sending that update to all systems that can be reached through the Ethernet3 interface:

```
ISO-IGRP: Originating level 2 periodic update
ISO-IGRP: Advertise dest: 0001
ISO-IGRP: Sending update on interface: Ethernet3
```

The following lines indicate that the router received an update from NSAP 3333.3333.3333 on Ethernet3. This update indicated the area the router at this NSAP could reach.

```
ISO-IGRP: Received update from 3333.3333.3333 (Ethernet3)
ISO-IGRP: Opcode: area
```

The following lines indicate that the router received an update advertising that the source of that update can reach area 1 with a metric of 1100. A station opcode indicates that the update included system addresses.

```
ISO-IGRP: Received level 2 adv for 0001 metric 1100
ISO-IGRP: Opcode: station
```

debug clns packet

Use the **debug clns packet** EXEC command to display information about packet receipt and forwarding to the next interface. The **no** form of this command disables debugging output.

[no] debug clns packet

Sample Display

Figure 2-48 shows sample **debug clns packet** output.

Figure 2-48 Sample Debug CLNS Packet Output

```
router# debug clns packet

CLNS: Forwarding packet size 157
      from 47.0023.0001.0000.0000.0003.0001.1920.3614.3002.00 STUPI-RBS
      to 47.0005.80ff.ef00.0000.0001.5940.1600.8906.4017.00
      via 1600.8906.4017 (Ethernet0 0000.0c00.bda8)
CLNS: Echo PDU received on Ethernet0 from 4
      7.0005.80ff.ef00.0000.0001.5940.1600.8906.4017.00!
CLNS: Sending from 47.0005.80ff.ef00.0000.0001.5940.1600.8906.4023.00 to
      47.0005.80ff.ef00.0000.0001.5940.1600.8906.4017.00
      via 1600.8906.4017 (Ethernet0 0000.0c00.bda8)
```

Explanations for individual lines of output from Figure 2-48 follow.

In the following lines, the first line indicates that a Connectionless Network Service (CLNS) packet of size 157 bytes is being forwarded. The second line indicates the network service access point (NSAP) and system name of the source of the packet. The third line indicates the destination NSAP for this packet. The fourth line indicates the next-hop system ID, interface, and SNPA of the router interface used to forward this packet.

```
CLNS: Forwarding packet size 157
      from 47.0023.0001.0000.0000.0003.0001.1920.3614.3002.00 STUPI-RBS
      to 47.0005.80ff.ef00.0000.0001.5940.1600.8906.4017.00
      via 1600.8906.4017 (Ethernet0 0000.0c00.bda8)
```

In the following lines, the first line indicates that the router received an Echo PDU on the specified interface from the source NSAP. The second line indicates which source NSAP is used to send a CLNS packet to the destination NSAP, as shown on the third line. The fourth line indicates the next-hop system ID, interface, and SNPA of the router interface used to forward this packet.

```
CLNS: Echo PDU received on Ethernet0 from
      47.0005.80ff.ef00.0000.0001.5940.1600.8906.4017.00!
CLNS: Sending from 47.0005.80ff.ef00.0000.0001.5940.1600.8906.4023.00 to
      47.0005.80ff.ef00.0000.0001.5940.1600.8906.4017.00
      via 1600.8906.4017 (Ethernet0 0000.0c00.bda8)
```

debug clns routing

Use the **debug clns routing** EXEC command to display debugging information for all Connectionless Network Service (CLNS) routing cache updates and activities involving the CLNS routing table. The **no** form of this command disables debugging output.

[no] debug clns routing

Sample Display

Figure 2-49 shows sample **debug clns routing** output.

Figure 2-49 Sample Debug CLNS Routing Output

```
router# debug clns routing

CLNS-RT: cache increment:17
CLNS-RT: Add 47.0023.0001.0000.0000.0003.0001 to prefix table, next hop 1920.3614.3002
CLNS-RT: Aging cache entry for: 47.0023.0001.0000.0000.0003.0001.1920.3614.3002.06
CLNS-RT: Deleting cache entry for: 47.0023.0001.0000.0000.0003.0001.1920.3614.3002.06
```

Explanations for individual lines of output from Figure 2-49 follow.

The following line indicates that a change to the routing table has resulted in an addition to the fast-switching cache:

```
CLNS-RT: cache increment:17
```

The following line indicates that a specific prefix route was added to the routing table, and indicates the next-hop system ID to that prefix route. In other words, when the router receives a packet with the prefix 47.0023.0001.0000.0000.0003.0001 in that packet's destination address, it forwards that packet to the router with the MAC address 1920.3614.3002.

```
CLNS-RT: Add 47.0023.0001.0000.0000.0003.0001 to prefix table, next hop 1920.3614.3002
```

The following lines indicate that the fast-switching cache entry for a certain network service access point (NSAP) has been invalidated and then deleted:

```
CLNS-RT: Aging cache entry for: 47.0023.0001.0000.0000.0003.0001.1920.3614.3002.06
CLNS-RT: Deleting cache entry for: 47.0023.0001.0000.0000.0003.0001.1920.3614.3002.06
```

debug cls message

Use the **debug cls message** EXEC command to display information about Cisco Link Services (CLS) messages. The **no** form of this command disables debugging output.

[no] debug cls message

Usage Guidelines

The **debug cls message** command displays the primitives (state), selector, header length, and data size.

Sample Display

Figure 2-50 shows sample **debug cls message** output. For example, *CLS-->DLU* indicates the direction of the flow that is described by the status. From CLS to DLU, a request was established to the connection end point. The header length is 48 bytes, and the data size is 104 bytes.

The status possibilities include the following: enabled, disabled, request open station, open station, close station, activate SA, deactivate SAP, XID, XID station, connect station, signal station, connect, disconnect, connected, data, flow, unnumbered data, modify SAP, test, activate ring, deactivate ring, test station, and unnumbered data station.

Figure 2-50 Sample Debug CLS Message Output

```
router# debug cls message

(FRAS Daemon:CLS-->DLU):
  ID_STN.Ind to uSAP: 0x607044C4 sel: LLC hlen: 40, dlen: 54
(FRAS Daemon:CLS-->DLU):
  ID_STN.Ind to uSAP: 0x6071B054 sel: LLC hlen: 40, dlen: 46
(FRAS Daemon:DLU-->SAP):
  REQ_OPNSTN.Reg to pSAP: 0x608021F4 sel: LLC hlen: 48, dlen: 104
(FRAS Daemon:CLS-->DLU):
  REQ_OPNSTN.Cfm(NO_REMOTE_STN) to uCEP: 0x607FFE84 sel: LLC hlen: 48, dlen: 104
```

Related Commands

debug fras error
debug fras message
debug fras state

debug cls vdlc

Use the **debug cls vdlc** EXEC command to display information about Cisco Link Services (CLS) Virtual Data Link Control (VDLC). The **no** form of this command disables debugging output.

[no] debug cls vdlc

Usage Guidelines



Caution Use the **debug cls vdlc** command with caution because it can generate a significant amount of output.

The **debug cls message** command displays primitive state transitions, selector, and source and destination media access control (MAC) and service access points (SAPs).

Also use the **show cls** command to display additional information on CLS VDLC.

Sample Displays

The following messages are sample **debug cls vdlc** output. In the following scenario, the SNA service point—also called *native service point (NSP)*—is setting up two connections through VDLC and data link switching (DLSw): one from NSP to VDLC and one from DLSw to VDLC. VDLC's task is to join the two.

The NSP initiates a connection from 4000.05d2.0001 as follows:

```
VDLC: Req Open Stn Req PSap 0x7ACE00, port 0x79DF98
      4000.05d2.0001(0C)->4000.1060.1000(04)
```

In the next message, VDLC sends a test station request to DLSw for destination address 4000.1060.1000.

```
VDLC: Send UFrame E3: 4000.05d2.0001(0C)->4000.1060.1000(00)
```

In the next two messages, DLSw replies with test station response, and NSP goes to a half-open state. NSP is waiting for the DLSw connection to VDLC.

```
VDLC: Sap to Sap TEST_STN_RSP VSap 0x7B68C0 4000.1060.1000(00)->4000.05d2.0001(0C)
VDLC: 4000.05d2.0001(0C)->4000.1060.1000(04): VDLC_OPENING->VDLC_HALF_OPEN
```

The NSP sends an exchange identification (XID) and changes state as follows:

```
VDLC: 4000.05d2.0001(0C)->4000.1060.1000(04): VDLC_HALF_OPEN->VDLC_XID_RSP_PENDING
VDLC: CEP to SAP ID_REQ 4000.05d2.0001(0C)->4000.1060.1000(04) via bridging SAP (DLSw)
```

In the next several messages, DLSw initiates its connection, which matches the half-open connection with NSP.

```
VDLC: Req Open Stn Req PSap 0x7B68C0, port 0x7992A0
      4000.1060.1000(04)->4000.05d2.0001(0C)
VDLC: two-way connection established
VDLC: 4000.1060.1000(04)->4000.05d2.0001(0C): VDLC_IDLE->VDLC_OPEN
```

In the following messages, DLSw sends an XID response, and NSP's connection goes from the state XID Response Pending to Open. The XID exchange follows:

```
VDLC: CEP to CEP ID_RSP 4000.1060.1000(04)->4000.05d2.0001(0C)
VDLC: 4000.05d2.0001(0C)->4000.1060.1000(04): VDLC_XID_RSP_PENDING->VDLC_OPEN
```

```

V DLC: 4000.05d2.0001(0C)->4000.1060.1000(04): V DLC_OPEN->V DLC_XID_RSP_PENDING
V DLC: CEP to CEP ID_REQ 4000.05d2.0001(0C)->4000.1060.1000(04)
V DLC: CEP to CEP ID_RSP 4000.1060.1000(04)->4000.05d2.0001(0C)
V DLC: 4000.05d2.0001(0C)->4000.1060.1000(04): V DLC_XID_RSP_PENDING->V DLC_OPEN
V DLC: 4000.05d2.0001(0C)->4000.1060.1000(04): V DLC_OPEN->V DLC_XID_RSP_PENDING
V DLC: CEP to CEP ID_REQ 4000.05d2.0001(0C)->4000.1060.1000(04)
V DLC: CEP to CEP ID_RSP 4000.1060.1000(04)->4000.05d2.0001(0C)
V DLC: 4000.05d2.0001(0C)->4000.1060.1000(04): V DLC_XID_RSP_PENDING->V DLC_OPEN
V DLC: 4000.05d2.0001(0C)->4000.1060.1000(04): V DLC_OPEN->V DLC_XID_RSP_PENDING
V DLC: CEP to CEP ID_REQ 4000.05d2.0001(0C)->4000.1060.1000(04)
V DLC: CEP to CEP ID_RSP 4000.1060.1000(04)->4000.05d2.0001(0C)
V DLC: 4000.05d2.0001(0C)->4000.1060.1000(04): V DLC_XID_RSP_PENDING->V DLC_OPEN
V DLC: 4000.05d2.0001(0C)->4000.1060.1000(04): V DLC_OPEN->V DLC_XID_RSP_PENDING
V DLC: CEP to CEP ID_REQ 4000.05d2.0001(0C)->4000.1060.1000(04)

```

When DL Sw is ready to connect, the front-end processor (FEP) sends a set asynchronous balanced mode extended (SABME) command as follows:

```

V DLC: CEP to CEP CONNECT_REQ 4000.1060.1000(04)->4000.05d2.0001(0C)
V DLC: 4000.05d2.0001(0C)->4000.1060.1000(04): V DLC_XID_RSP_PENDING->V DLC_OPEN

```

In the following messages, NSP accepts the connection and sends an unnumbered acknowledgment (UA) to the FEP:

```

V DLC: CEP to CEP CONNECT_RSP 4000.05d2.0001(0C)->4000.1060.1000(04)
V DLC: FlowReq QUENCH OFF 4000.1060.1000(04)->4000.05d2.0001(0C)

```

The following messages show the data flow:

```

V DLC: DATA 4000.1060.1000(04)->4000.05d2.0001(0C)
V DLC: DATA 4000.05d2.0001(0C)->4000.1060.1000(04)
...
V DLC: DATA 4000.1060.1000(04)->4000.05d2.0001(0C)
V DLC: DATA 4000.05d2.0001(0C)->4000.1060.1000(04)

```

Related Commands

- debug cls message**
- debug dlsw core message**

debug compress

Use the **debug compress** EXEC command to display compression information. The **no** form of this command disables debugging output.

[no] debug compress

Sample Display

Figure 2-51 shows sample **debug compress** output.

Figure 2-51 Sample Debug Compress Output

```
router# debug compress
DECOMPRESS xmt_paks 5 rcv_sync 5
      COMPRESS xmt_paks 10 version 1
      COMPRESS xmt_paks 11 version 1
DECOMPRESS xmt_paks 6 rcv_sync 6
      COMPRESS xmt_paks 12 version 1
      COMPRESS xmt_paks 13 version 1
DECOMPRESS xmt_paks 7 rcv_sync 7
      COMPRESS xmt_paks 14 version 1
      COMPRESS xmt_paks 15 version 1
```

Table 2-22 describes significant fields shown in Figure 2-51.

Table 2-22 Debug Compress Field Descriptions

Field	Description
COMPRESS xmt_paks	The sequence count of this frame is modulo 256 (except zero only occurs on initialization). This value is part of the compression header sent with each frame.
DECOMPRESS xmt_paks	The sequence count in the compression header received with this frame.
DECOMPRESS rcv_sync	The received internal sequence count, which is verified against the DECOMPRESS xmt_paks count. If these counts do not match, a Link Access Procedure, Balanced (LAPB) reset will occur. On LAPB reset, a compression reinitialization occurs. Compression reinitialization initializes the dictionaries and xmt_paks and rcv_sync counts.

debug confmodem

Use the **debug confmodem** EXEC command to display information associated with the discovery and configuration of the modem attached to the router. The **no** form of this command disables debugging output.

[no] debug confmodem

Usage Guidelines

The **debug confmodem** command is used in debugging configurations that use the **modem autoconfig** command.

Sample Display

Figure 2-52 shows sample **debug confmodem** output. In the first three lines, the router is searching for a speed at which it can communicate with the modem. The remaining lines show the actual sending of the modem command.

Figure 2-52 Sample Debug Configuration Modem Output

```
router# debug confmodem

TTY4:detection speed(115200) response -----
TTY4:detection speed(57600) response -----
TTY4:detection speed(38400) response ---OK---
TTY4:Modem command: --AT&F&C1&D2S180=3S190=1S0=1--
TTY4: Modem configuration succeeded
TTY4: Done with modem configuration
```

debug cpp event

Use the **debug cpp event** EXEC command to display general Combinet Proprietary Protocol (CPP) events. The **no** form of this command disables debugging output.

[no] **debug cpp event**

Usage Guidelines

The CPP protocol allows a router to engage in negotiation over an ISDN B channel to establish connections with a Combinet bridge.

The **debug cpp event** command displays events such as CPP sequencing, group creation, and keepalives.

Sample Displays

One or more of the messages shown in Table 2-23 appear when you use the **debug cpp event** command. Each message begins with the short name of the interface the event occurred on (for example, *SERIAL0:1* or *BRI0:1*) and might contain one or more packet sequence numbers or remote site names.

Table 2-23 Debug CPP Event Messages

Message	Description
BRI0:1: negotiation complete	The call was set up on the interface (in this example, BRI0:1).
BRI0:1: negotiation timed out	The call timed out.
BRI0:1: sending negotiation packet	The negotiation packet was sent to set up the call.
BRI0:1: out of sequence packet - got 10, range 1 8	A packet was received that was out of sequence. The first number displayed in the message is the sequence number received and the following numbers are the range of valid sequence numbers.
BRI0:1: Sequence timer expired - Lost 11 Trying sequence 12	The timer expired before the packet was received. The first number displayed in the message is the sequence number of the packet that was lost, and the second number is the next sequence number.
BRI0:1: Line Integrity Violation	This message occurs when the router fails to maintain keepalives.
BRI0:1: create cpp group ber19 destroyed cpp group ber19	This message occurs when a dialer group is created on the remote site (in this example, <i>ber19</i>).

Related Commands

debug cpp negotiation

debug cpp packet

debug cpp negotiation

Use the **debug cpp negotiation** EXEC command to display Combinet Proprietary Protocol (CPP) negotiation events. The **no** form of this command disables debugging output.

[no] debug cpp negotiation

Usage Guidelines

The CPP protocol allows a router to engage in negotiation over an ISDN B channel to establish connections with a Combinet bridge.

The **debug cpp negotiation** command displays events such as the type of packet and packet size being sent.

Sample Display

Figure 2-53 shows sample **debug cpp negotiation** output. In this example, a sample connection is shown.

Figure 2-53 Sample Debug CPP Negotiation Output

```
router# debug cpp negotiation
%LINK-3-UPDOWN: Interface BRI0: B-Channel 2, changed state to down
%LINK-3-UPDOWN: Interface BRI0, changed state to up
%SYS-5-CONFIG_I: Configured from console by console
%LINK-3-UPDOWN: Interface BRI0: B-Channel 1, changed state to up
BR0:1:(I) NEG packet - len 77
  attempting proto:2
  ether id:0040.f902.c7b4
  port 1 number:5559876
  port 2 number:5559876
  origination port:1
  remote name:berl9
  password is correct
```

Table 2-24 shows describes the fields and messages shown in Figure 2-53.

Table 2-24 Debug CPP Negotiation Field Descriptions

Field	Description
BR0:1 (I) NEG packet - len 77	Interface name, packet type, and packet size.
attempting proto:	CPP protocol type.
ether id:	Ethernet address of the destination router.
port 1 number:	ISDN phone number of remote B channel #1.
port 2 number:	ISDN phone number of remote B channel #2.
origination port:	B channel 1 or 2 called.
remote name:	Remote site name to which this call is connecting.
password is correct	Password is accepted so the connection is established.

Related Commands

debug cpp event
debug cpp packet

debug cpp packet

Use the **debug cpp packet** EXEC command to display Combinet Proprietary Protocol (CPP) packets. The **no** form of this command disables debugging output.

[no] debug cpp packet

Usage Guidelines

The CPP protocol allows a router to engage in negotiation over an ISDN B channel to establish connections with a Combinet bridge.

The **debug cpp packet** command displays the hexadecimal values of the packets.

Sample Display

Figure 2-54 shows sample **debug cpp packet** output. This example shows the interface name, packet type, packet size, and the hexadecimal values of the packet.

Figure 2-54 Sample Debug CPP Packet Output

```
router# debug cpp packet

BR0:1:input packet - len 60
00 00 00 00 00 00 00 00 40 F9 02 C7 B4 08 0. !6 00 01
08 00 06 04 00 02 00 40 F9 02 C7 B4 83 6C A1 02!!!
Success rate is 80 percent (4/5), round-trip min/avg/max = 64/66/68 ms
BR0:1 output packet - len 116
06 00 00 40 F9 02 C7 B4 00 00 0C 3E 12 3A 08 00
45 00 00 64 00 01 00 00 FF 01 72 BB 83 6C A1 01
```

Related Commands

debug cpp event

debug cpp negotiation

debug crypto key-exchange

Use the **debug crypto key-exchange** EXEC command to show Digital Signature Standard (DSS) public key exchange messages. The **no** form of this command disables debugging output.

[no] debug crypto key-exchange

Usage Guidelines

Encryption and authentication are provided by a software service on the router called a *crypto engine*. The crypto engine performs authentication through DSS public and private keys when a connection is set up. DSS is a means of sending a “signature” at the end of a message that positively identifies the author of the message. The signature cannot be forged or duplicated by others, so whoever received a message with a DSS signature knows exactly who sent the message.

If the process of exchanging DSS public keys with a peer router by means of the **config crypto key-exchange** command is not successful, try to exchange DSS public keys again after enabling the **debug crypto key-exchange** command to help you diagnose the problem.

Sample Displays

Figure 2-55 and Figure 2-56 show sample **debug crypto key-exchange** output. Figure 2-55 shows output from the initiating router in a key exchange. Figure 2-56 shows output from the passive router in a key exchange. The number of bytes received should match the number of bytes sent from the initiating side, although the number of messages can be different.

Figure 2-55 Sample Debug Crypto Key-Exchange Output—Initiating Router

```
router# debug crypto key-exchange
CRYPTO-KE: Sent 4 bytes.
CRYPTO-KE: Sent 2 bytes.
CRYPTO-KE: Sent 2 bytes.
CRYPTO-KE: Sent 2 bytes.
CRYPTO-KE: Sent 64 bytes.
```

Figure 2-56 Sample Debug Crypto Key-Exchange Output—Passive Router

```
router# debug crypto key-exchange
CRYPTO-KE: Received 4 bytes.
CRYPTO-KE: Received 2 bytes.
CRYPTO-KE: Received 2 bytes.
CRYPTO-KE: Received 2 bytes.
CRYPTO-KE: Received 49 bytes.
CRYPTO-KE: Received 15 bytes.
```

Related Command

debug crypto sesgmt

debug crypto sesmgmt

Use the **debug crypto sesmgmt** EXEC command to show connection setup messages and their flow through the router. The **no** form of this command disables debugging output.

[no] debug crypto sesmgmt

Usage Guidelines

Encryption and authentication are provided by a software service on the router called a *crypto engine*. The crypto engine performs authentication through DSS public and private keys when a connection is set up. DSS is a means of sending a “signature” at the end of a message that positively identifies the author of the message. The signature cannot be forged or duplicated by others, so whoever received a message with a DSS signature knows exactly who sent the message.

When connections are not completing, use the **debug crypto sesmgmt** command to follow the progress of connection messages as a first step in diagnosing the problem. You see a record of each connection message as the router discovers it, and can track its progress through the necessary signing, verifying, and encryption session setup operations. Other significant connection setup events, such as the pregeneration of Diffie-Hellman public numbers, are also shown. For information on Diffie-Hellman public numbers, refer to the *Security Configuration Guide*.

Also use the **show crypto connections** command to display additional information on connections.

Sample Displays

Figure 2-57 and Figure 2-58 show sample **debug crypto sesmgmt** output. Figure 2-57 shows messages from a router that initiates a successful connection. Figure 2-58 shows the messages from a router that receives a connection.

Figure 2-57 Sample Debug Crypto Sesmgmt Output—Initiating Router

```
router# debug crypto sesmgmt
CRYPTO: Dequeued a message: Inititate_Connection
CRYPTO: DH gen phase 1 status for conn_id 2 slot 0:OK
CRYPTO: Signing done. Status:OK
CRYPTO: ICMP message sent: s=172.21.114.163, d=172.21.114.162
CRYPTO-SDU: send_nnc_req: NNC Echo Request sent
CRYPTO: Dequeued a message: CRM
CRYPTO: DH gen phase 2 status for conn_id 2 slot 0:OK
CRYPTO: Verify done. Status=OK
CRYPTO: Signing done. Status:OK
CRYPTO: ICMP message sent: s=172.21.114.163, d=172.21.114.162
CRYPTO-SDU: recv_nnc_rpy: NNC Echo Confirm sent
CRYPTO: Create encryption key for conn_id 2 slot 0:OK
CRYPTO: Replacing -2 in crypto maps with 2 (slot 0)
```

Figure 2-58 Sample Debug Crypto Sesmgmt Output—Receiving Router

```
router# debug crypto sesmgmt
CRYPTO: Dequeued a message: CIM
CRYPTO: Verify done. Status=OK
CRYPTO: DH gen phase 1 status for conn_id 1 slot 0:OK
CRYPTO: DH gen phase 2 status for conn_id 1 slot 0:OK
CRYPTO: Signing done. Status:OK
CRYPTO: ICMP message sent: s=172.21.114.162, d=172.21.114.163
CRYPTO-SDU: act_on_nnc_req: NNC Echo Reply sent
```

```
CRYPTO: Create encryption key for conn_id 1 slot 0:OK  
CRYPTO: Replacing -2 in crypto maps with 1 (slot 0)  
CRYPTO: Dequeued a message: CCM  
CRYPTO: Verify done. Status=OK
```

Related Command

debug crypto key-exchange

debug decnet adj

Use the **debug decnet adj** EXEC command to display debugging information on DECnet adjacencies. The **no** form of this command disables debugging output.

[no] debug decnet adj

Sample Display

Figure 2-59 shows sample **debug decnet adj** output.

Figure 2-59 Sample Debug DECnet Adj Output

```
router# debug decnet adj
DECnet adjacencies debugging is on
router#
DNET-ADJ: Level 1 hello from 1.3
DNET-ADJ: sending hellos
DNET-ADJ: Sending hellos to all routers on interface Ethernet0, blksize 1498
DNET-ADJ: Level 1 hello from 1.3
DNET-ADJ: 1.5 adjacency initializing
DNET-ADJ: sending triggered hellos
DNET-ADJ: Sending hellos to all routers on interface Ethernet0, blksize 1498
DNET-ADJ: Level 1 hello from 1.3
DNET-ADJ: 1.5 adjacency up
DNET-ADJ: Level 1 hello from 1.5
DNET-ADJ: 1.5 adjacency down, listener timeout
```

Explanations for representative lines of output in Figure 2-59 follow.

The following line indicates that the router is sending hellos to all routers on this segment, which in this case is Ethernet 0:

```
DNET-ADJ: Sending hellos to all routers on interface Ethernet0, blksize 1498
```

The following line indicates that the router has heard a hello from address 1.5 and is creating an adjacency entry in its table. The initial state of this adjacency will be *initializing*.

```
DNET-ADJ: 1.5 adjacency initializing
```

The following line indicates that the router is sending an unscheduled (triggered) hello as a result of some event, such as new adjacency being heard:

```
DNET-ADJ: sending triggered hellos
```

The following line indicates that the adjacency with 1.5 is now up, or active:

```
DNET-ADJ: 1.5 adjacency up
```

The following line indicates that the adjacency with 1.5 has timed out, because no hello has been heard from adjacency 1.5 in the time interval originally specified in the hello from 1.5:

```
DNET-ADJ: 1.5 adjacency down, listener timeout
```

The following line indicates that the router is sending an unscheduled hello, as a result of some event, such as the adjacency state changing:

```
DNET-ADJ: hello update triggered by state changed in dn_add_adjacency
```

debug decnet connects

Use the **debug decnet connects** EXEC command to display debugging information of all connect packets that are filtered (permitted or denied) by DECnet access lists. The **no** form of this command disables debugging output.

[no] debug decnet connects

Usage Guidelines

When you use connect packet filtering, it may be helpful to use the **decnet access-group** configuration command to apply the following basic access list:

```
access-list 300 permit 0.0 63.1023 eq any
```

You can then log all connect packets transmitted on interfaces to which you applied this list, in order to determine those elements on which your connect packets must be filtered.

Note Packet password and account information is not logged in the **debug decnet connects** message, nor is it displayed by the **show access** EXEC command. If you specify **password** or **account** information in your access list, they can be viewed by anyone with access to the configuration of the router.

Sample Display

Figure 2-60 shows sample **debug decnet connects** output.

Figure 2-60 Sample Debug DECnet Connects Output

```
router# debug decnet connects
DNET-CON: list 300 item #2 matched src=19.403 dst=19.309 on Ethernet0: permitted
srcname="RICK" srcuic=[0,017]
dstobj=42 id="USER"
```

Table 2-25 describes significant fields shown in Figure 2-60.

Table 2-25 Debug DECnet Connects Field Descriptions

Field	Description
DNET-CON:	Indicates that this is a debug decnet connects packet
list 300 item #2 matched	Indicates that a packet matched the second item in access list 300
src = 19.403	Indicates the source DECnet address for the packet
dst = 19.309	Indicates the destination DECnet address for the packet
on Ethernet0:	Indicates the router interface on which the access list filtering the packet was applied
permitted	Indicates that the access list permitted the packet
srcname = "RICK"	Indicates the originator user of the packet
srcuic = [0,017]	Indicates the source UIC of the packet

Table 2-25 Debug DECnet Connects Field Descriptions (Continued)

Field	Description
dstobj = 42	Indicates that DECnet object 42 is the destination
id="USER"	Indicates the access user

debug decnet events

Use the **debug decnet events** EXEC command to display debugging information on DECnet events. The **no** form of this command disables debugging output.

[no] debug decnet events

Sample Display

Figure 2-61 shows sample **debug decnet events** output.

Figure 2-61 Sample Debug DECnet Events Output

```
router# debug decnet events  
  
DNET: Hello from area 50 rejected - exceeded 'max area' parameter (45)  
DNET: Hello from area 50 rejected - exceeded 'max area' parameter (45)
```

Explanations for representative lines of output in Figure 2-61 follow.

The following line indicates that the router received a hello from a router whose area was greater than the max-area parameter with which this router was configured:

```
DNET: Hello from area 50 rejected - exceeded 'max area' parameter (45)
```

The following line indicates that the router received a hello from a router whose node ID was greater than the max-node parameter with which this router was configured:

```
DNET: Hello from node 1002 rejected - exceeded 'max node' parameter (1000)
```

debug decnet packet

Use the **debug decnet packet** EXEC command to display debugging information on DECnet packet events. The **no** form of this command disables debugging output.

[no] debug decnet packet

Sample Display

Figure 2-62 shows sample **debug decnet packet** output.

Figure 2-62 Sample Debug DECnet Packet Output

```
router# debug decnet packet

DNET-PKT: src 1.4 dst 1.5 sending to PHASEV
DNET-PKT: Packet fwded from 1.4 to 1.5, via 1.5, snpa 0000.3080.cf90, TokenRing0
```

Explanations for individual lines of output from Figure 2-62 follow.

The following line indicates that the router is sending a converted packet addressed to node 1.5 to Phase V:

```
DNET-PKT: src 1.4 dst 1.5 sending to PHASEV
```

The following line indicates that the router forwarded a packet from node 1.4 to node 1.5. The packet is being sent to the next hop of 1.5 whose subnetwork point of attachment (MAC address) on that interface is 0000.3080.cf90.

```
DNET-PKT: Packet fwded from 1.4 to 1.5, via 1.5, snpa 0000.3080.cf90, TokenRing0
```

debug decnet routing

Use the **debug decnet routing** EXEC command to display all DECnet routing-related events occurring at the router. The **no** form of this command disables debugging output.

[no] debug decnet routing

Sample Display

Figure 2-63 shows sample **debug decnet routing** output.

Figure 2-63 Sample Debug DECnet Routing Output

```
router# debug decnet routing

DNET-RT: Received level 1 routing from 1.3 on Ethernet0 at 1:16:34
DNET-RT: Sending routes
DNET-RT: Sending normal routing updates on Ethernet0
DNET-RT: Sending level 1 routing updates on interface Ethernet0
DNET-RT: Level1 routes from 1.5 on Ethernet0: entry for node 5 created
DNET-RT: route update triggered by after split route pointers in dn_rt_input
DNET-RT: Received level 1 routing from 1.5 on Ethernet 0 at 1:18:35
DNET-RT: Sending L1 triggered routes
DNET-RT: Sending L1 triggered routing updates on Ethernet0
DNET-RT: removing route to node 5
```

Explanations for individual lines of output from Figure 2-63 follow.

The following line indicates that the router has received a level 1 update on interface Ethernet 0:

```
DNET-RT: Received level 1 routing from 1.3 on Ethernet0 at 1:16:34
```

The following line indicates that the router is sending its scheduled updates on interface Ethernet 0:

```
DNET-RT: Sending normal routing updates on Ethernet0
```

The following line indicates that the route will send an unscheduled update on this interface as a result of some event. In this case, the unscheduled update is a result of a new entry created in the interface's routing table.

```
DNET-RT: route update triggered by after split route pointers in dn_rt_input
```

The following line indicates that the router sent the unscheduled update on Ethernet 0:

```
DNET-RT: Sending L1 triggered routes
DNET-RT: Sending L1 triggered routing updates on Ethernet0
```

The following line indicates that the router removed the entry for node 5 because the adjacency with node 5 timed out, or the route to node 5 through a next-hop router went away:

```
DNET-RT: removing route to node 5
```

debug dialer events

Use the **debug dialer events** EXEC command to display debugging information about the packets received on a dialer interface. The **no** form of this command disables debugging output.

[no] debug dialer events

Sample Displays

When DDR is enabled on the interface, information concerning the cause of any call (called the *Dialing cause*) is displayed. The following line of output for an IP packet lists the name of the DDR interface and the source and destination addresses of the packet:

```
Dialing cause: Serial0: ip (s=172.16.1.111 d=172.16.2.22)
```

The following line of output for a bridged packet lists the DDR interface and the type of packet (in hexadecimal). For information on these packet types, see the “Ethernet Type Codes” appendix of the *Bridging and IBM Networking Command Reference* publication.

```
Dialing cause: Serial1: Bridge (0x6005)
```

Most messages are self-explanatory; however, messages that may need some explanation are described in Table 2-26.

Table 2-26 General Debug Dialer Events Message Descriptions

Message	Description
Dialer0: Already <i>xxx</i> call(s) in progress on Dialer0, dialing not allowed	This message occurs when the number of calls in progress (<i>xxx</i>) exceeds the maximum number of calls set on the interface.
Dialer0: No free dialer - starting fast idle timer	This message occurs when all the lines in the interface or rotary group are busy and a packet is waiting to be sent to the destination.
BRI0: rotary group to <i>xxx</i> overloaded (<i>yyy</i>)	This message occurs when the number dialer (<i>xxx</i>) exceeds the load set on the interface (<i>yyy</i>).
BRI0: authenticated host <i>xxx</i> with no matching dialer profile	This message occurs when no dialer profile matches <i>xxx</i> , the remote host's CHAP name or remote name.
BRI0: authenticated host <i>xxx</i> with no matching dialer map	This message occurs when no dialer map matches <i>xxx</i> , the remote host's CHAP name or remote name.
BRI0: Can't place call, verify configuration	This message occurs when you have not set the dialer string or dialer pool on an interface.

Table 2-27 describes the messages that the **debug dialer events** command can generate for a serial interface used as a V.25*bis* dialer for dial-on-demand routing (DDR).

Table 2-27 Debug Dialer Events Message Descriptions for DDR

Message	Description
Serial 0: Dialer result = xxxxxxxx	This message displays the result returned from the V.25bis dialer. It is useful in debugging if calls are failing. On some hardware platforms, this message cannot be displayed due to hardware limitations. Possible values for the xxxxxxxx variable depend on the V.25bis device with which the router is communicating.
Serial 0: No dialer string defined. Dialing cannot occur.	This message is displayed when a packet is received that should cause a call to be placed. However, there is no dialer string configured, so dialing cannot occur. This message usually indicates a configuration problem.
Serial 0: Attempting to dial xxxxxxxx	This message indicates that a packet has been received that passes the dial-on-demand access lists. That packet causes phone number xxxxxxxx to be dialed.
Serial 0: Unable to dial xxxxxxxx	This message is displayed if for some reason the phone call to xxxxxxxx cannot be placed. This failure might be due to a lack of memory, full output queues, or other problems.
Serial 0: disconnecting call	This message is displayed when the router hangs up a call.
Serial 0: idle timeout Serial 0: re-enable timeout Serial 0: wait for carrier timeout	One of these three messages is displayed when a dialer timer expires. These messages are mostly informational, but are useful for debugging a disconnected call or call failure.

Related Command
debug dialer packets

debug dialer packets

Use the **debug dialer packets** EXEC command to display debugging information about the packets received on a dialer interface. The **no** form of this command disables debugging output.

[no] debug dialer packets

Usage Guidelines

Most debug dialer packet messages are self-explanatory.

Sample Display

Figure 2-64 shows sample **debug dialer packets** output. The following message shows the interface type, the type of packet (protocol) being sent, the source and destination addresses, the size of the packet, and the default action for the packet (in this example, *permit*).

Figure 2-64 Sample Debug Dialer Packets Output

```
router# debug dialer packets  
BRI0: ip (s=10.1.1.8, d=10.1.1.1), 100 bytes, interesting (ip PERMIT)
```

Related Command

debug dialer events

debug dlsw

Use the **debug dlsw** EXEC command to enable debugging of data-link switching (DLSw). The **no** form of this command disables debugging output.

```
[no] debug dlsw [core [circuit-number | flow-control | messages | state | xid] | local-circuit |
peer | reachability [[error | verbose] [sna | netbios]]]
```

Syntax Description

core	(Optional) Enables debugging output for DLSw core events.
<i>circuit-number</i>	(Optional) Specifies the circuit for which you want core debugging output to reduce the of output.
flow-control	(Optional) Enables debugging output for congestion in the WAN or at the remote end station.
messages	(Optional) Enables debugging output of core messages—specific packets received by DLSw either from one of its peers or from a local medium via the Cisco Link Services Interface (CLSI).
state	(Optional) Enables debugging output for state changes on the circuit.
xid	(Optional) Enables debugging output for the exchange identification (XID) state machine.
local-circuit	(Optional) Enables debugging output for circuits performing local conversion. Local conversion occurs when both the input and output data-link connections are on the same local peer and no remote peer exists.
peer	(Optional) Enables debugging output for peer events.
reachability	(Optional) Enables debugging output for reachability events (explorer traffic). If no options are specified, event-level information is displayed for all protocols.
error verbose	(Optional) Specifies how much reachability information you want displayed. The verbose keyword displays everything, including errors and events, while the error keyword displays error information only. If no option is specified, event-level information is displayed.
sna netbios	(Optional) Specifies that reachability information be displayed for only SNA or NetBIOS protocols. If no option is specified, information for all protocols is displayed.

Usage Guidelines

When you specify no optional keywords, the **debug dlsw** command enables all available DLSw debugging output.

Normally you only need to use the **error** or **verbose** option of the **debug dlsw reachability** command to help identify problems. The **error** option is recommended for use by customers and provides a subset of the messages from the normal event-level debugging. The **verbose** option provides a very detailed view of what is going on and is typically used only by service personnel.

To reduce the amount of debug information displayed, use the **sna** or **netbios** options with the **debug dlsw reachability** command if you know that you have an SNA or NetBIOS problem.

The DLSw core is the engine that is responsible for the establishment and maintenance of remote circuits. If possible, specifying the index of the specific circuit you want to debug reduces the amount of output displayed. However, if you want to watch a circuit initially come up, do not use the *circuit-number* option with the **core** keyword.

The **core flow-control** option provides information about congestion in the WAN or at the remote end station. In these cases, DLSw sends Receiver Not Ready (RNR) frames on its local circuits, throttling data traffic on established sessions and giving the congestion an opportunity to clear.

The **core state** option allows you to see when the circuit changes state. This capability is especially useful for determining why a session cannot be established or why a session is being disconnected.

The **core XID** option allows you to track the XID state machine. The router tracks XID commands and responses used in negotiations between end stations before establishing a session.

Sample Displays

The following sections show and explain some of the typical DLSw debug messages you might see when using the **debug dlsw peer** and **debug dlsw reachability** commands.

Sample Debug DLSW Peer Messages

The following messages occur when a CUR_ex (CANUREACH explorer) frame is received from other peers, and the peer statements or the **promiscuous** keyword have not been enabled so that the router is misconfigured:

```
22:42:44: DLSw: Not promiscuous - Rej conn from 172.20.96.1(2065)
22:42:51: DLSw: Not promiscuous - Rej conn from 172.20.99.1(2065)
```

In the following messages, the router sends out a keepalive message every 30 seconds to keep the peer connected. If three keepalive messages are missed, the peer is torn down. These messages are display only if keepalives are enabled (by default, keepalives are disabled).

```
22:44:03: DLSw: Keepalive Request sent to peer 172.20.98.1(2065) (168243148)
22:44:03: DLSw: Keepalive Response from peer 172.20.98.1(2065) (168243176)
22:44:34: DLSw: Keepalive Request sent to peer 172.20.98.1(2065) (168274148)
22:44:34: DLSw: Keepalive Response from peer 172.20.98.1(2065) (168274172)
```

The following peer debug messages indicate that the local peer is disconnecting from the specified remote peer due to missed peer keepalives:

```
0:03:24: DLSw: keepalive failure for peer on interface Serial0
0:03:24: DLSw: action_d(): for peer on interface Serial0
0:03:24: DLSw: DIRECT aborting connection for peer on interface Serial0
0:03:24: DLSw: peer on interface Serial0, old state CONNECT, new state DISCONN
```

The following peer debug messages result from an attempt to connect to an IP address that does not have DLSw enabled. The local router attempts to connect in 30-second intervals.

```
23:13:22: action_a() attempting to connect peer 172.20.100.1(2065)
23:13:22: DLSw: CONN: peer 172.20.100.1 open failed, rejected [9]
23:13:22: action_a() retries: 8 next conn time: 861232504
23:13:52: action_a() attempting to connect peer 172.20.100.1(2065)
```

```
23:13:52: DLSw: CONN: peer 172.20.100.1 open failed, rejected [9]
23:13:52: action_a() retries: 9 next conn time: 861292536
```

The following peer debug messages indicates a remote-peer statement is missing on the router at address 172.20.100.1 to which the connection attempt is sent:

```
23:14:52: action_a() attempting to connect peer 172.20.100.1(2065)
23:14:52: DLSw: action_a(): Write pipe opened for peer 172.20.100.1(2065)
23:14:52: DLSw: peer 172.20.100.1(2065), old state DISCONN, new state WAIT_RD
23:14:52: DLSw: dlsw_tcpd_fini() closing connection for peer 172.20.100.1
23:14:52: DLSw: action_d(): for peer 172.20.100.1(2065)
23:14:52: DLSw: aborting tcp connection for peer 172.20.100.1(2065)
23:14:52: DLSw: peer 172.20.100.1(2065), old state WAIT_RD, new state DISCONN
```

The following messages show a normal opening:

```
23:16:37: action_a() attempting to connect peer 172.20.100.1(2065)
23:16:37: DLSw: action_a(): Write pipe opened for peer 172.20.100.1(2065)
23:16:37: DLSw: peer 172.20.100.1(2065), old state DISCONN, new state WAIT_RD
23:16:37: DLSw: passive open 172.20.100.1(17762) -> 2065
23:16:37: DLSw: action_c(): for peer 172.20.100.1(2065)
23:16:37: DLSw: peer 172.20.100.1(2065), old state WAIT_RD, new state CAP_EXG
23:16:37: DLSw: peer 172.20.100.1(2065) conn_start_time set to 861397784
23:16:37: DLSw: CapExId Msg sent to peer 172.20.100.1(2065)
23:16:37: DLSw: Recv CapExId Msg from peer 172.20.100.1(2065)
23:16:37: DLSw: Pos CapExResp sent to peer 172.20.100.1(2065)
23:16:37: DLSw: action_e(): for peer 172.20.100.1(2065)
23:16:37: DLSw: Recv CapExPosRsp Msg from peer 172.20.100.1(2065)
23:16:37: DLSw: action_e(): for peer 172.20.100.1(2065)
23:16:37: DLSw: peer 172.20.100.1(2065), old state CAP_EXG, new state CONNECT
23:16:37: DLSw: dlsw_tcpd_fini() closing write pipe for peer 172.20.100.1
23:16:37: DLSw: action_g(): for peer 172.20.100.1(2065)
23:16:37: DLSw: closing write pipe tcp connection for peer 172.20.100.1(2065)
23:16:38: DLSw: peer_act_on_capabilities() for peer 172.20.100.1(2065)
```

The following two messages show that an information frame is passing through:

```
DLSw: dlsw_tr2fct() lmac:c000.a400.0000 rmac:0800.5a29.75fe ls:5 rs:4 i:34
DLSw: dlsw_tr2fct() lmac:c000.a400.0000 rmac:0800.5a29.75fe ls:4 rs:4 i:34
```

Sample Debug DLSw Reachability Messages

The messages in this section are based on the following items:

- Reachability is stored in cache. DLSw+ maintains two reachability caches: one for media access control (MAC) addresses and one for NetBIOS names. Depending on how long entries have been in the cache, they are either fresh or stale.
- If a router has a fresh entry in the cache for a certain resource, it answers a locate request for that resource without verifying that it is still available. A locate request is typically a TEST frame for MAC addresses, or a FIND_NAME_QUERY for NetBIOS.
- If a router has a stale entry in the cache for a certain resource, it verifies that the entry is still valid before answering a locate request for the resource by sending a frame to the resource's last known location and awaits a resource. If the entry is a REMOTE entry, the router sends a CUR_ex frame to the remote peer to verify. If the entry is a LOCAL entry, it sends either a TEST frame or a NetBIOS FIND_NAME_QUERY on the appropriate local port.
- By default, all reachability cache entries remain fresh for 4 minutes after they are learned. For MAC addresses, you can change this time with the **dlsw timer sna-verify-interval** command. For NetBIOS names, you can change this with the **dlsw timer netbios-verify-interval** command.

- By default, all reachability cache entries age out of the cache 16 minutes after they are learned. For MAC addresses, you can change this time with the **dlsw timer sna-cache-timeout** command. For NetBIOS names, you can change the time with the **dlsw timer netbios-cache-timeout** command.

Table 2-28 describes the debug output indicating that the dlsw router received an SSP message that is flow controlled and should be counted against the sender's window.

```
Dec 6 11:26:49: CSM: Received SSP CUR csex flags = 80, mac 4000.90b1.26cf,
The csex flags = 80 means that this is an CUR_ex (explorer).
Dec 5 10:48:33: DLSw: 1620175180 decr r - s:27 so:0 r:27 ro:0
```

Table 2-28 Debug Output Descriptions

Field	Description
dec r	Decrement received count
s	This dlsw router's granted units for the circuit
so	0=This dlsw router does not owe a flow control acknowledgement. 1=This router owes a flow control acknowledgement.
r	The partner's number of granted units for the circuit.
ro	Indicates whether the partner owes flow control acknowledgement

The following message shows that DLSw is sending an I frame to a LAN:

```
Dec 5 10:48:33: DISP Sent : CLSI Msg : DATA.Reg dlen: 1086
```

The following message shows that DLSw received the I frame from the LAN:

```
Dec 5 10:48:35: DLSW Received-disp : CLSI Msg : DATA.Ind dlen: 4
```

The following messages show that the reachability cache is cleared:

```
Router# clear dlsw rea

23:44:11: CSM: Clearing CSM cache
23:44:11: CSM: delete local mac cache for port 0
23:44:11: CSM: delete local name cache for port 0
23:44:11: CSM: delete remote mac cache for peer 0
23:44:11: CSM: delete remote name cash dlsw rea
```

The next group of messages show that the DLSw reachability cache is added, and that a name query is perform from the router Marian:

```
23:45:11: CSM: core_to_csm CLSI_MSG_PROC - port_id 5EFBB4
23:45:11: CSM: 0800.5a30.7a9b passes local mac excl. filter
23:45:11: CSM: update local cache for mac 0800.5a30.7a9b, port 5EFBB4
23:45:11: CSM: update local cache for name MARIAN , port 5EFBB4
23:45:11: CSM: Received CLS_UDATA_STN from Core
23:45:11: CSM: Received netbios frame type A
23:45:11: CSM: Processing Name Query
23:45:11: CSM: Netbios Name Query: ws_status = 6
23:45:11: CSM: Write to peer 0 ok.
```

```

23:45:11: CSM: Freeing clsi message
23:45:11: CSM: core_to_csm CLSI_MSG_PROC - port_id 658AB4
23:45:11: CSM: 0800.5a30.7a9b passes local mac excl. filter
23:45:11: CSM: update local cache for mac 0800.5a30.7a9b, port 658AB4
23:45:11: CSM: update local cache for name MARIAN , port 658AB4
23:45:11: CSM: Received CLS_UDATA_STN from Core
23:45:11: CSM: Received netbios frame type A
23:45:11: CSM: Processing Name Query
23:45:11: CSM: Netbios Name Query: ws_status = 5
23:45:11: CSM: DLXNR_PEND match found.... drop name query
23:45:11: CSM: Freeing clsi message
23:45:12: CSM: core_to_csm CLSI_MSG_PROC - port_id 5EFBB4
23:45:12: CSM: 0800.5a30.7a9b passes local mac excl. filter
23:45:12: CSM: update local cache for mac 0800.5a30.7a9b, port 5EFBB4
23:45:12: CSM: update local cache for name MARIAN , port 5EFBB4
23:45:12: CSM: Received CLS_UDATA_STN from Core
23:45:12: CSM: Received netbios frame type A
23:45:12: CSM: Processing Name Query
23:45:12: CSM: Netbios Name Query: ws_status = 5
23:45:12: CSM: DLXNR_PEND match found.... drop name query
23:45:12: CSM: Freeing clsi message
23:45:12: CSM: core_to_csm CLSI_MSG_PROC - port_id 658AB4
23:45:12: CSM: 0800.5a30.7a9b passes local mac excl. filter
23:45:12: CSM: update local cache for mac 0800.5a30.7a9b, port 658AB4
23:45:12: CSM: update local cache for name MARIAN , port 658AB4
23:45:12: CSM: Received CLS_UDATA_STN from Core
23:45:12: CSM: Received netbios frame type A
23:45:12: CSM: Processing Name Query
23:45:12: CSM: Netbios Name Query: ws_status = 5
23:45:12: CSM: DLXNR_PEND match found.... drop name query
23:45:12: CSM: Freeing clsi message
23:45:12: CSM: core_to_csm CLSI_MSG_PROC - port_id 5EFBB4
23:45:12: CSM: 0800.5a30.7a9b passes local mac excl. filter
23:45:12: CSM: update local cache for mac 0800.5a30.7a9b, port 5EFBB4
23:45:12: CSM: update local cache for name MARIAN , port 5EFBB4
23:45:12: CSM: Received CLS_UDATA_STN from Core
23:45:12: CSM: Received netbios frame type A
23:45:12: CSM: Processing Name Query
23:45:12: CSM: Netbios Name Query: ws_status = 5
23:45:12: CSM: DLXNR_PEND match found.... drop name query
23:45:12: CSM: Freeing clsi message
23:45:12: CSM: core_to_csm CLSI_MSG_PROC - port_id 658AB4
23:45:12: CSM: 0800.5a30.7a9b passes local mac excl. filter
23:45:12: CSM: update local cache for mac 0800.5a30.7a9b, port 658AB4
23:45:12: CSM: update local cache for name MARIAN , port 658AB4
23:45:12: CSM: Received CLS_UDATA_STN from Core
23:45:12: CSM: Received netbios frame type A
23:45:12: CSM: Processing Name Query
23:45:12: CSM: Netbios Name Query: ws_status = 5
23:45:12: CSM: DLXNR_PEND match found.... drop name query
23:45:12: CSM: Freeing clsi message
23:45:18: CSM: Deleting Reachability cache
23:45:18: CSM: Deleting DLX NR pending record...
23:45:38: CSM: core_to_csm CLSI_MSG_PROC - port_id 5EFBB4
23:45:38: CSM: 0800.5a30.7a9b passes local mac excl. filter
23:45:38: CSM: update local cache for mac 0800.5a30.7a9b, port 5EFBB4
23:45:38: CSM: update local cache for name MARIAN , port 5EFBB4
23:45:38: CSM: Received CLS_UDATA_STN from Core
23:45:38: CSM: Received netbios frame type 8
23:45:38: CSM: Write to peer 0 ok.
23:45:38: CSM: Freeing clsi message
23:45:38: CSM: core_to_csm CLSI_MSG_PROC - port_id 658AB4
23:45:38: CSM: 0800.5a30.7a9b passes local mac excl. filter
23:45:38: CSM: update local cache for mac 0800.5a30.7a9b, port 658AB4
23:45:38: CSM: update local cache for name MARIAN , port 658AB4

```

```

23:45:38: CSM: Received CLS_UDATA_STN from Core
23:45:38: CSM: Received netbios frame type 8
23:45:38: CSM: Write to peer 0 ok.
23:45:38: CSM: Freeing clsi message

```

The following messages show that Marian is added to the network:

```

23:45:38: CSM: core_to_csm CLSI_MSG_PROC - port_id 5EFBB4
23:45:38: CSM: 0800.5a30.7a9b passes local mac excl. filter
23:45:38: CSM: update local cache for mac 0800.5a30.7a9b, port 5EFBB4
23:45:38: CSM: update local cache for name MARIAN , port 5EFBB4
23:45:38: CSM: Received CLS_UDATA_STN from Core
23:45:38: CSM: Received netbios frame type 8
23:45:38: CSM: Write to peer 0 ok.
23:45:38: CSM: Freeing clsi message
23:45:38: CSM: core_to_csm CLSI_MSG_PROC - port_id 658AB4
23:45:38: CSM: 0800.5a30.7a9b passes local mac excl. filter
23:45:38: CSM: update local cache for mac 0800.5a30.7a9b, port 658AB4
23:45:38: CSM: update local cache for name MARIAN , port 658AB4
23:45:38: CSM: Received CLS_UDATA_STN from Core
23:45:38: CSM: Received netbios frame type 8
23:45:38: CSM: Write to peer 0 ok.
23:45:38: CSM: Freeing clsi message

```

In the next group of messages, an attempt to add the router Ginger on the Ethernet is made:

```

0:07:44: CSM: core_to_csm CLSI_MSG_PROC - port_id 658AB4
0:07:44: CSM: 0004.f545.24e6 passes local mac excl. filter
0:07:44: CSM: update local cache for mac 0004.f545.24e6, port 658AB4
0:07:44: CSM: update local cache for name GINGER , port 658AB4
0:07:44: CSM: Received CLS_UDATA_STN from Core
0:07:44: CSM: Received netbios frame type 8
0:07:44: CSM: Write to peer 0 ok.

```

In the following example, the output from the **show dlsw reachability** command indicates that Ginger is on the Ethernet interface and Marian is on the Token Ring interface:

```

G41#sh dlsw rea

DLSw MAC address reachability cache list
Mac Addr      status      Loc.      peer/port      rif
0004.f545.24e6 FOUND      LOCAL    P007-S000     --no rif--
0800.5a30.7a9b FOUND      LOCAL    P000-S000     06C0.0621.7D00
                                P007-S000     F0F8.0006.A6FC.005F.F100.0000.0000.0000

DLSw NetBIOS Name reachability cache list
NetBIOS Name  status      Loc.      peer/port      rif
GINGER        FOUND      LOCAL    P007-S000     --no rif--
MARIAN        FOUND      LOCAL    P000-S000     06C0.0621.7D00
                                P007-S000     --no rif--

```

debug dspu activation

Use the **debug dspu activation** EXEC command to display information on downstream physical unit (DSPU) activation. The **no** form of this command disables debugging output.

[no] debug dspu activation *[name]*

Syntax Description

name (Optional) A host or PU name designation.

Usage Guidelines

The **debug dspu activation** command displays all DSPU activation traffic. To restrict the output to a specific host or physical unit (PU), include the host or PU name argument. You cannot turn off debugging output for an individual PU if that PU has not been named in the **debug dspu activation** command.

Sample Display

Figure 2-65 shows sample **debug dspu activation** output. Not all intermediate numbers are shown for the “activated” and “deactivated” logical unit (LU) address ranges.

Figure 2-65 Sample Debug DSPU Activation Output

```
router# debug dspu activation
DSPU: LS HOST3745 connected
DSPU: PU HOST3745 activated
DSPU: LU HOST3745-2 activated
DSPU: LU HOST3745-3 activated
...
DSPU: LU HOST3745-253 activated
DSPU: LU HOST3745-254 activated

DSPU: LU HOST3745-2 deactivated
DSPU: LU HOST3745-3 deactivated
...
DSPU: LU HOST3745-253 deactivated
DSPU: LU HOST3745-254 deactivated
DSPU: LS HOST3745 disconnected
DSPU: PU HOST3745 deactivated
```

Table 2-29 describes significant fields in the output shown in Figure 2-65.

Table 2-29 Debug DSPU Activation Field Descriptions

Field	Description
DSPU	Downstream PU debug message.
LS	A link station (LS) event triggered the message.
PU	A PU event triggered the message.
LU	A logical unit (LU) event triggered the message.
HOST3745	Host name or PU name.

Table 2-29 **Debug DSPU Activation Field Descriptions (Continued)**

Field	Description
HOST3745-253	Host name or PU name and the LU address, separated by a colon.
connected activated disconnected deactivated	Event that occurred to trigger the message.

Related Commands**debug dspu packet****debug dspu state****debug dspu trace**

debug dspu packet

Use the **debug dspu packet** EXEC command to display information on downstream physical unit (DSPU) packet. The **no** form of this command disables debugging output.

[no] debug dspu packet [*name*]

Syntax Description

name (Optional) A host or PU name designation.

Usage Guidelines

The **debug dspu packet** command displays all DSPU packet data flowing through the router. To restrict the output to a specific host or PU, include the host or PU *name* argument. You cannot turn off debugging output for an individual PU if that PU has not been named in the **debug dspu packet** command.

Sample Display

Figure 2-66 shows sample **debug dspu packet** output.

Figure 2-66 Sample Debug DSPU Packet Output

```
router# debug dspu packet

DSPU: Rx: PU HOST3745 data length 12 data:
      2D0003002BE16B80 000D0201
DSPU: Tx: PU HOST3745 data length 25 data:
      2D0000032BE1EB80 000D020100850000 000C060000010000 00
DSPU: Rx: PU HOST3745 data length 12 data:
      2D0004002BE26B80 000D0201
DSPU: Tx: PU HOST3745 data length 25 data:
      2D0000042BE2EB80 000D020100850000 000C060000010000 00
```

Table 2-30 describes significant fields in the output shown in Figure 2-66.

Table 2-30 Debug DSPU Packet Field Descriptions

Field	Description
DSPU: Rx:	Received frame (packet) from the remote PU to the router PU.
DSPU: Tx:	Transmitted frame (packet) from the router PU to the remote PU.
PU HOST3745	Host name or PU associated with the transmit or receive.
data length 12 data:	Number of bytes of data, followed by up to 128 bytes of displayed data.

Related Commands

debug dspu activation
debug dspu state
debug dspu trace

debug dspu state

Use the **debug dspu state** EXEC command to display information on downstream physical unit (DSPU) finite state machine (FSM) state changes. The **no** form of this command disables debugging output.

```
[no] debug dspu state [name]
```

Syntax Description

name (Optional) A host or PU name designation.

Usage Guidelines

Use the **debug dspu state** command to display only the FSM state changes. To see all FSM activity, use the **debug dspu trace command**. You cannot turn off debugging output for an individual PU if that PU has not been named in the **debug dspu state** command.

Sample Display

Figure 2-67 shows sample **debug dspu state** output. Not all intermediate numbers are shown for the “activated” and “deactivated” logical unit (LU) address ranges.

Figure 2-67 Sample Debug DSPU State Output

```
router# debug dspu state
DSPU: LS HOST3745: input=StartLs, Reset -> PendConOut
DSPU: LS HOST3745: input=ReqOpn.Cnf, PendConOut -> Xid
DSPU: LS HOST3745: input=Connect.Ind, Xid -> ConnIn
DSPU: LS HOST3745: input=Connected.Ind, ConnIn -> Connected
DSPU: PU HOST3745: input=Actpu, Reset -> Active
DSPU: LU HOST3745-2: input=uActlu, Reset -> upLuActive
DSPU: LU HOST3745-3: input=uActlu, Reset -> upLuActive
...
DSPU: LU HOST3745-253: input=uActlu, Reset -> upLuActive
DSPU: LU HOST3745-254: input=uActlu, Reset -> upLuActive

DSPU: LS HOST3745: input=PuStopped, Connected -> PendDisc
DSPU: LS HOST3745: input=Disc.Cnf, PendDisc -> PendClose
DSPU: LS HOST3745: input=Close.Cnf, PendClose -> Reset
DSPU: PU HOST3745: input=T2ResetPu, Active -> Reset
DSPU: LU HOST3745-2: input=uStopLu, upLuActive -> Reset
DSPU: LU HOST3745-3: input=uStopLu, upLuActive -> Reset
...
DSPU: LU HOST3745-253: input=uStopLu, upLuActive -> Reset
DSPU: LU HOST3745-254: input=uStopLu, upLuActive -> Reset
```

Table 2-31 describes significant fields in the output shown in Figure 2-67.

Table 2-31 Debug DSPU State Field Descriptions

Field	Description
DSPU	Downstream PU debug message.
LS	A link station (LS) event triggered the message.

Table 2-31 Debug DSPU State Field Descriptions (Continued)

Field	Description
PU	A PU event triggered the message.
LU	A logical unit (LU) event triggered the message.
HOST3745-253	Host name or PU name and LU address.
input= <i>input</i> ,	The input received by the FSM.
<i>previous-state</i> , -> <i>current-state</i>	The previous state and current new state as seen by the FSM.

Related Commands

- debug dspu activation**
- debug dspu packet**
- debug dspu trace**

debug dspu trace

Use the **debug dspu trace** EXEC command to display information on downstream physical unit (DSPU) trace activity, which includes all finite state machine (FSM) activity. The **no** form of this command disables debugging output.

```
[no] debug dspu trace [name]
```

Syntax Description

name (Optional) A host or PU name designation.

Usage Guidelines

Use the **debug dspu trace** command to display all FSM state changes. To see FSM state changes only, use the **debug dspu state** command. You cannot turn off debugging output for an individual PU if that PU has not been named in the **debug dspu trace** command.

Sample Display

Figure 2-68 shows sample **debug dspu trace** output.

Figure 2-68 Sample Debug DSPU Trace Output

```
router# debug dspu trace

DSPU: LS HOST3745 input = 0 ->(1,a1)
DSPU: LS HOST3745 input = 5 ->(5,a6)
DSPU: LS HOST3745 input = 7 ->(5,a9)
DSPU: LS HOST3745 input = 9 ->(5,a28)
DSPU: LU HOST3745-2 in:0 s:0->(2,a1)
DSPU: LS HOST3745 input = 19 ->(8,a20)
DSPU: LS HOST3745 input = 18 ->(8,a17)
DSPU: LU HOST3745-3 in:0 s:0->(2,a1)
DSPU: LS HOST3745 input = 19 ->(8,a20)
DSPU: LS HOST3745 input = 18 ->(8,a17)
DSPU: LU HOST3745-252 in:0 s:0->(2,a1)
DSPU: LS HOST3745 input = 19 ->(8,a20)
DSPU: LS HOST3745 input = 18 ->(8,a17)
DSPU: LU HOST3745-253 in:0 s:0->(2,a1)
DSPU: LS HOST3745 input = 19 ->(8,a20)
DSPU: LS HOST3745 input = 18 ->(8,a17)
DSPU: LU HOST3745-254 in:0 s:0->(2,a1)
DSPU: LS HOST3745 input = 19 ->(8,a20)
```

Table 2-32 describes significant fields in the output shown in Figure 2-68.

Table 2-32 Debug DSPU Trace Field Descriptions

Field	Description
7:23:57	Time stamp.
DSPU	Downstream PU debug message.
LS	A link station (LS) event triggered the message.
PU	A PU event triggered the message.

Table 2-32 Debug DSPU Trace Field Descriptions (Continued)

Field	Description
LU	A logical unit (LU) event triggered the message.
HOST3745-253	Host name or PU name and LU address.
in: <i>input s:state ->(new-state, action)</i>	String describing the following: <i>input</i> - LU FSM input <i>state</i> - Current FSM state <i>new-state</i> - New FSM state <i>action</i> - FSM action
input= <i>input ->(new-state, action)</i>	String describing the following: <i>input</i> - PU or LS FSM input <i>new-state</i> - New PU or LS FSM state <i>action</i> - PU or LS FSM action

Related Commands

- debug dspu activation**
- debug dspu packet**
- debug dspu state**

debug eigrp fsm

Use the **debug eigrp fsm** EXEC command to display debugging information about Enhanced IGRP feasible successor metrics (FSM). The **no** form of this command disables debugging output.

[no] debug eigrp fsm

Usage Guidelines

This command helps you observe Enhanced IGRP feasible successor activity and to determine whether route updates are being installed and deleted by the routing process.

Sample Display

Figure 2-69 shows sample **debug eigrp fsm** output.

Figure 2-69 Sample Debug EIGRP FSM Output

```
router# debug eigrp fsm

DUAL: dual_rcvupdate(): 172.25.166.0 255.255.255.0 via 0.0.0.0 metric 750080/0
DUAL: Find FS for dest 172.25.166.0 255.255.255.0. FD is 4294967295, RD is 42949
67295 found
DUAL: RT installed 172.25.166.0 255.255.255.0 via 0.0.0.0
DUAL: dual_rcvupdate(): 192.168.4.0 255.255.255.0 via 0.0.0.0 metric 4294967295/
4294967295
DUAL: Find FS for dest 192.168.4.0 255.255.255.0. FD is 2249216, RD is 2249216
DUAL: 0.0.0.0 metric 4294967295/4294967295not found Dmin is 4294967295
DUAL: Dest 192.168.4.0 255.255.255.0 not entering active state.
DUAL: Removing dest 192.168.4.0 255.255.255.0, nexthop 0.0.0.0
DUAL: No routes. Flushing dest 192.168.4.0 255.255.255.0
```

Explanations for individual lines of output from Figure 2-69 follow.

In the first line of Figure 2-69, DUAL stands for Diffusing Update ALgorithm. It is the basic mechanism within Enhanced IGRP that makes the routing decisions. The next three fields are the Internet address and mask of the destination network and the address through which the update was received. The metric field shows the metric stored in the routing table and the metric advertised by the neighbor sending the information. “Metric... inaccessible” usually means that the neighbor router no longer has a route to the destination, or the destination is in holddown.

In the following output, Enhanced IGRP is attempting to find a feasible successor for the destination. Feasible successors are part of the DUAL loop avoidance methods. The FD field contains more loop avoidance state information. The RD field is the reported distance, which is the metric used in update, query or reply packets.

The indented line with the “not found” message means a feasible successor (FS) was not found for 192.168.4.0 and EIGRP must start a diffusing computation. This means it begins to actively probe (sends query packets about destination 192.168.4.0) the network looking for alternate paths to 192.164.4.0.

```
DUAL: Find FS for dest 192.168.4.0 255.255.255.0. FD is 2249216, RD is 2249216
DUAL: 0.0.0.0 metric 4294967295/4294967295not found Dmin is 4294967295
```

The following output indicates the route DUAL successfully installed into the routing table.

```
DUAL: RT installed 172.25.166.0 255.255.255.0 via 0.0.0.0
```

The following output shows that no routes were discovered to the destination and the route information is being removed from the topology table.

```
DUAL: Dest 192.168.4.0 255.255.255.0 not entering active state.  
DUAL: Removing dest 192.168.4.0 255.255.255.0, nexthop 0.0.0.0  
DUAL: No routes. Flushing dest 192.168.4.0 255.255.255.0
```

debug eigrp packet

Use the **debug eigrp packet EXEC** command to display general debugging information. The **no** form of this command disables debugging output.

[no] debug eigrp packet

Usage Guidelines

If a communication session is closing when it should not be, an end-to-end connection problem can be the cause. The **debug eigrp packet** command is useful for analyzing the messages traveling between the local and remote hosts.

Sample Display

Figure 2-70 shows sample **debug eigrp packet** output.

Figure 2-70 Sample Debug EIGRP Packet Output

```
router# debug eigrp packet

EIGRP: Sending HELLO on Ethernet0/1
      AS 109, Flags 0x0, Seq 0, Ack 0
EIGRP: Sending HELLO on Ethernet0/1
      AS 109, Flags 0x0, Seq 0, Ack 0
EIGRP: Sending HELLO on Ethernet0/1
      AS 109, Flags 0x0, Seq 0, Ack 0
EIGRP: Received UPDATE on Ethernet0/1 from 192.195.78.24,
      AS 109, Flags 0x1, Seq 1, Ack 0
EIGRP: Sending HELLO/ACK on Ethernet0/1 to 192.195.78.24,
      AS 109, Flags 0x0, Seq 0, Ack 1
EIGRP: Sending HELLO/ACK on Ethernet0/1 to 192.195.78.24,
      AS 109, Flags 0x0, Seq 0, Ack 1
EIGRP: Received UPDATE on Ethernet0/1 from 192.195.78.24,
      AS 109, Flags 0x0, Seq 2, Ack 0
```

The output shows transmission and receipt of Enhanced IGRP packets. These packet types may be HELLO, UPDATE, REQUEST, QUERY, or REPLY packets. The sequence and acknowledgment numbers used by the Enhanced IGRP reliable transport algorithm are shown in the output. Where applicable, the network layer address of the neighboring router is also included.

Table 2-33 describes significant fields in the output shown in Figure 2-70.

Table 2-33 Debug EIGRP Packet Field Descriptions

Field	Description
EIGRP:	An Enhanced IGRP packet.
AS <i>n</i>	Autonomous system number.
Flags <i>nxn</i>	A flag of 1 means the sending router is indicating to the receiving router that this is the first packet it has sent to the receiver. A flag of 2 is a multicast that should be conditionally received by routers that have the conditionally-receive (CR) bit set. This bit gets set when the sender of the multicast has previously sent a sequence packet explicitly telling it to set the CR bit.

Table 2-33 Debug EIGRP Packet Field Descriptions (Continued)

Field	Description
HELLO	The hello packets are the neighbor discovery packets. They are used to determine whether neighbors are still alive. As long as neighbors receive the hello packets the router is sending, the neighbors validate the router and any routing information sent. If neighbors lose the hello packets, the receiving neighbors invalidate any routing information previously sent. Neighbors also transmit hello packets.

debug fddi smt-packets

Use the **debug fddi smt-packets** EXEC command to display information about Station Management (SMT) frames received by the router. The **no** form of this command disables debugging output.

[no] debug fddi smt-packets

Sample Display

Figure 2-71 shows sample **debug fddi smt-packets** output. In this example, an SMT frame has been output by Fiber Distributed Data Interface (FDDI) 1/0. The SMT frame is a next station addressing (NSA) neighbor information frame (NIF) request frame with the parameters as shown.

Figure 2-71 Sample Debug FDDI SMT Output

```
router# debug fddi smt-packets
SMT O: Fddi1/0, FC=NSA, DA=ffff.ffff.ffff, SA=00c0.eeee.be04,
  class=NIF, type=Request, vers=1, station_id=00c0.eeee.be04, len=40
  - code 1, len 8 -- 000000016850043F
  - code 2, len 4 -- 00010200
  - code 3, len 4 -- 00003100
  - code 200B, len 8 -- 0000000100000000
```

Table 2-34 describes the fields in the output shown in Figure 2-71.

Table 2-34 Debug FDDI SMT-Packets Field Descriptions

Field	Description
SMT O	An SMT frame was transmitted from the interface FDDI 1/0. Also, SMT I indicates an SMT frame was received on the interface FDDI 1/0.
Fddi1/0	The interface associated with the frame.
FC	Frame control byte in the media access control (MAC) header.
DA, SA	Destination and source addresses in FDDI form.
class	Frame class. Values can be echo frame (ECF), neighbor information frame (NIF), parameter management frame (PMF), request denied frame (RDF), status information frame (SIF), and status report frame (SRF).
type	Frame type. Values can be Request, Response, and Announce.
vers	Version identification. Values can be 1 or 2.
station_id	Station identification.
len	Packet size.
code 1, len 8 -- 000000016850043F	Parameter type X'0001—upstream neighbor address (UNA), parameter length in bytes, and parameter value. SMT parameters are described in the SMT specification ANSI X3T9.

debug frame-relay

Use the **debug frame-relay** EXEC command to display debugging information about the packets that are received on a Frame Relay interface. The **no** form of this command disables debugging output.

[no] debug frame-relay

Usage Guidelines

This command helps you analyze the packets that have been received. However, because the **debug frame-relay** command generates a lot of output, only use it when traffic on the Frame Relay network is less than 25 packets per second.

To analyze the packets that have been *sent* on a Frame Relay interface, use the **debug frame-relay packet** command.

Sample Display

Figure 2-72 shows sample **debug frame-relay** output.

Figure 2-72 Sample Debug Frame-Relay Output

```
router# debug frame-relay

Serial0(i): dlci 500(0x7C41), pkt type 0x809B, datagramsize 24
Serial1(i): dlci 1023(0xFCF1), pkt type 0x309, datagramsize 13
Serial0(i): dlci 500(0x7C41), pkt type 0x809B, datagramsize 24
Serial1(i): dlci 1023(0xFCF1), pkt type 0x309, datagramsize 13
Serial0(i): dlci 500(0x7C41), pkt type 0x809B, datagramsize 24
```

Table 2-35 describes significant fields shown in Figure 2-72.

Table 2-35 Debug Frame-Relay Field Descriptions

Field	Description
Serial0(i):	Indicates that the Serial0 interface has received this Frame Relay datagram as input.
dlci 500(0x7C41)	Indicates the value of the data link connection identifier (DLCI) for this packet in decimal (and q922). In this case, 500 has been configured as the multicast DLCI.
pkt type 0x809B	Indicates the packet type code. Possible supported signaling message codes follow: 0x308—Signaling message; valid only with a DLCI of 0. 0x309—LMI message; valid only with a DLCI of 1023

Table 2-35 Debug Frame-Relay Field Descriptions (Continued)

Field	Description
pkt type 0x809B (continued)	<p>Possible supported Ethernet type codes follow:</p> <p>0x0201—IP on 3MB net</p> <p>0x0201—Xerox ARP on 10MB nets</p> <p>0xCC—RFC 1294 (only for IP)</p> <p>0x0600—XNS</p> <p>0x0800—IP on 10 MB net</p> <p>0x0806—IP ARP</p> <p>0x0808—Frame Relay ARP</p> <p>0x0BAD—VINES IP</p> <p>0x0BAE—VINES loopback protocol</p> <p>0x0BAF—VINES Echo</p> <p>Possible HDLC type codes follow:</p> <p>0x6001—DEC MOP booting protocol</p> <p>0x6002—DEC MOP console protocol</p> <p>0x6003—DECnet Phase IV on Ethernet</p> <p>0x6004—DEC LAT on Ethernet</p> <p>0x8005—HP Probe</p> <p>0x8035—RARP</p> <p>0x8038—DEC spanning tree</p> <p>0x809b—Apple EtherTalk</p> <p>0x80f3—AppleTalk ARP</p> <p>0x8019—Apollo domain</p> <p>0x80C4—VINES IP</p> <p>0x80C5— VINES ECHO</p> <p>0x8137—IPX</p> <p>0x9000—Ethernet loopback packet IP</p> <p>0x1A58— IPX, standard form</p> <p>0xFEFE—CLNS</p> <p>0xEFEF—ES-IS</p> <p>0x1998—Uncompressed TCP</p> <p>0x1999—Compressed TCP</p> <p>0x6558—Serial line bridging</p>
datagramsize 24	Indicates size of this datagram in bytes.

debug frame-relay callcontrol

Use the **debug frame-relay callcontrol** EXEC command to display Frame Relay Layer 3 (network layer) call control information. The **no** form of this command disables debugging output.

[no] debug frame-relay callcontrol

Usage Guidelines

The **debug frame-relay callcontrol** command is used specifically for observing FRF.4/Q.933 signaling messages and related state changes. The FRF.4/Q.933 specification describes a state machine for call control. The signaling code implements the state machine. The debug statements display the actual event and state combinations.

The Frame Relay switched virtual circuit (SVC) signaling subsystem is an independent software module. When used with the **debug frame-relay networklayerinterface** command, the **debug frame-relay callcontrol** command provides a better understanding of the call setup and teardown sequence. The **debug frame-relay networklayerinterface** command provides the details of the interactions between the signaling subsystem on the router and the Frame Relay subsystem.

Sample Display

The following state changes can be observed during a call setup on the calling party side. The **debug frame-relay networklayerinterface** command shows the following state changes or transitions:

```
STATE_NULL -> STATE_CALL_INITIATED -> STATE_CALL_PROCEEDING->STATE_ACTIVE
```

The following messages are samples of output generated during a call setup on the calling side:

```
6d20h: U0_SetupRequest: Serial0
6d20h: L3SDL: Ref: 1, Init: STATE_NULL, Rcvd: SETUP_REQUEST, Next: STATE_CALL_INITIATED
6d20h: U1_CallProceeding: Serial0
6d20h: L3SDL: Ref: 1, Init: STATE_CALL_INITIATED, Rcvd: MSG_CALL_PROCEEDING, Next:
STATE_CALL_PROCEEDING
6d20h: U3_Connect: Serial0
6d20h: L3SDL: Ref: 1, Init: STATE_CALL_PROCEEDING, Rcvd: MSG_CONNECT, Next: STATE_ACTIVE
6d20h:
```

The following messages are samples of output generated during a call setup on the called party side. Note the following state transitions as the call goes to the active state:

```
STATE_NULL -> STATE_CALL_PRESENT-> STATE_INCOMING_CALL_PROCEEDING->STATE_ACTIVE

1w4d: U0_Setup: Serial2/3
1w4d: L3SDL: Ref: 32769, Init: STATE_NULL, Rcvd: MSG_SETUP, Next: STATE_CALL_PRESENT
1w4d: L3SDL: Ref: 32769, Init: STATE_CALL_PRESENT, Rcvd: MSG_SETUP, Next:
STATE_INCOMING_CALL_PROC 1w4d: L3SDL: Ref: 32769, Init: STATE_INCOMING_CALL_PROC,
Rcvd: MSG_SETUP, Next: STATE_ACTIVE
```

Table 2-36 explains the possible call states.

Table 2-36 Frame Relay Switched Virtual Circuit (SVC) Call States

Call State	Description
Null	No call exists.
Call Initiated	User has requested the network to establish a call.

Table 2-36 Frame Relay Switched Virtual Circuit (SVC) Call States (Continued)

Call State	Description
Outgoing Call Proceeding	User has received confirmation from the network that the network has received all call information necessary to establish the call.
Call Present	User has received a request to establish a call but has not yet responded.
Incoming Call Proceeding	User has sent acknowledgment that all call information necessary to establish the call has been received (for an incoming call).
Active	On the called side, the network has indicated that the calling user has been awarded the call. On the calling side, the remote user has answered the call.
Disconnect Request	User has requested that the network clear the end-to-end call and is waiting for a response.
Disconnect Indication	User has received an invitation to disconnect the call because the network has disconnected the call.
Release Request	User has requested that the network release the call and is waiting for a response.

Related Commands**debug frame-relay****debug frame-relay networklayerinterface**

debug frame-relay events

Use the **debug frame-relay events** EXEC command to display debugging information about Frame Relay ARP replies on networks that support a multicast channel and use dynamic addressing. The **no** form of this command disables debugging output.

[no] debug frame-relay events

Usage Guidelines

This command is useful for identifying the cause of end-to-end connection problems during the installation of a Frame Relay network or node.

Note Because the **debug frame-relay events** command does not generate much output, you can use it at any time, even during periods of heavy traffic, without adversely affecting other users on the system.

Sample Display

Figure 2-73 shows sample **debug frame-relay events** output.

Figure 2-73 Sample Debug Frame-Relay Events Output

```
router# debug frame-relay events

Serial2(i): reply rcvd 172.16.170.26 126
Serial2(i): reply rcvd 172.16.170.28 128
Serial2(i): reply rcvd 172.16.170.34 134
Serial2(i): reply rcvd 172.16.170.38 144
Serial2(i): reply rcvd 172.16.170.41 228
Serial2(i): reply rcvd 172.16.170.65 325
```

As Figure 2-73 shows, **debug frame-relay events** returns one specific message type. The first line, for example, indicates that IP address 172.16.170.26 sent a Frame Relay ARP reply; this packet was received as input on the Serial2 interface. The last field (126) is the data link connection identifier (DLCI) to use when communicating with the responding router.

debug frame-relay informationelements

Use the **debug frame-relay informationelements** EXEC command to display information about Frame Relay Layer 3 (network layer) information element parsing and construction. The **no** form of this command disables debugging output.

[no] debug frame-relay informationelements

Usage Guidelines

Within the FRF.4/Q.933 signaling specification, messages are divided into subunits called information elements. Each information element defines parameters specific to the call. These parameters can be values configured on the router, or values requested from the network.

The **debug frame-relay informationelements** command shows the signaling message in hexadecimal. Use this command to determine parameters being requested and granted for a call.

Note Use the **debug frame-relay informationelements** command when the **debug frame-relay callcontrol** command offers no clues as to why calls are not being set up.

Note The **debug frame-relay informationelements** command displays a large amount of information in bytes. You must be familiar with FRF.4/Q.933 to decode the information contained within the debug output.

Sample Display

Figure 2-74 shows sample **debug frame-relay informationelements** output. In this example, each information element has a length associated with it. For those with odd-numbered lengths, only the specified bytes are valid, and the extra byte is invalid. For example, in the message “Call Ref, length: 3, 0x0200 0x0100,” only “02 00 01” is valid, the last “00” is invalid.

Figure 2-74 Sample Debug Frame-Relay Information Elements Output

```
1w0d# debug frame-relay informationelements
1w0d: Outgoing MSG_SETUP

1w0d: Dir: U --> N, Type: Prot Disc, length: 1, 0x0800
1w0d: Dir: U --> N, Type: Call Ref, length: 3, 0x0200 0x0100
1w0d: Dir: U --> N, Type: Message type, length: 1, 0x0500
1w0d: Dir: U --> N, Type: Bearer Capability, length: 5, 0x0403 0x88A0 0xCF00
1w0d: Dir: U --> N, Type: DLCI, length: 4, 0x1902 0x46A0
1w0d: Dir: U --> N, Type: Link Lyr Core, length: 27, 0x4819 0x090B 0x5C0B 0xDC0A
1w0d:           0x3140 0x31C0 0x0B21 0x4021
1w0d:           0xC00D 0x7518 0x7598 0x0E09
1w0d:           0x307D 0x8000
1w0d: Dir: U --> N, Type: Calling Party, length: 12, 0x6C0A 0x1380 0x3837 0x3635
1w0d:           0x3433 0x3231
1w0d: Dir: U --> N, Type: Calling Party Subaddr, length: 4, 0x6D02 0xA000
1w0d: Dir: U --> N, Type: Called Party, length: 11, 0x7009 0x9331 0x3233 0x3435
1w0d:           0x3637 0x386E
1w0d: Dir: U --> N, Type: Called Party Subaddr, length: 4, 0x7102 0xA000
1w0d: Dir: U --> N, Type: Low Lyr Comp, length: 5, 0x7C03 0x88A0 0xCE65
1w0d: Dir: U --> N, Type: User to User, length: 4, 0x7E02 0x0000
```

Table 2-37 explains the information elements in the example shown in Figure 2-74.

Table 2-37 Information Elements in a Setup Message

Information Element	Description
Prot Disc	Protocol discriminator.
Call Ref	Call reference.
Message Type	Message type such as <i>setup</i> , <i>connect</i> , and <i>call proceeding</i> .
Bearer Capability	Coding format such as data type and Layer 2 and Layer 3 protocols.
DLCI	Data-link connection identifier.
Link Lyr Core	Link layer core quality of service (QOS) requirements.
Calling Party	Type of source number (X121/E164) and the number.
Calling Party Subaddr	Subaddress that originated the call.
Called Party	Type of destination number (X121/E164) and the number.
Called Party Subaddr	Subaddress of the called party.
Low Lyr Comp	Coding format, data type, Layer 2 and Layer 3 protocols intended for the end user.
User to User	Information between end users.

Related Command

debug frame-relay callcontrol

debug frame-relay lapf

Use the **debug frame-relay lapf** EXEC command to display Frame Relay switched virtual circuit (SVC) Layer 2 information. The **no** form of this command disables debugging output.

[no] debug frame-relay lapf

Usage Guidelines

Use the **debug frame-relay lapf** command to troubleshoot the data-link control portion of Layer 2 that runs over data-link connection identifier (DLCI) 0. Use this command only if you have a problem bringing up Layer 2. You can use the **show interface serial** command to determine the status of Layer 2. If it shows a Link Access Procedure, Frame Relay (LAPF) state of down, Layer 2 has a problem.

Sample Displays

Figure 2-75 shows sample **debug frame-relay lapf** output. In this example, a line being brought up indicates an exchange of set asynchronous balanced mode extended (SAMBE) and unnumbered acknowledgment (UA) commands. A SABME is initiated by both sides, and a UA is the response. Until the SABME gets a UA response, the line is not declared to be up. The *p/f* value indicates the poll/final bit setting. *TX* means *send*, and *RX* means *receive*.

Figure 2-75 Sample Debug Frame-Relay LAPF Output—SABME-UA Exchange

```
1w0d# debug frame-relay lapf
1w0d: *LAPF Serial0 TX -> SABME Cmd p/f=1
1w0d: *LAPF Serial0 Enter state 5
1w0d: *LAPF Serial0 RX <- UA Rsp p/f=1
1w0d: *LAPF Serial0 lapf_ua_5
1w0d: *LAPF Serial0 Link up!
1w0d: *LAPF Serial0 RX <- SABME Cmd p/f=1
1w0d: *LAPF Serial0 lapf_sabme_78
1w0d: *LAPF Serial0 TX -> UA Rsp p/f=1
```

In the example shown in Figure 2-76, a line in an up LAPF state should see a steady exchange of RR (receiver ready) messages. *TX* means *send*, *RX* means *receive*, and *N(R)* indicates the receive sequence number.

Figure 2-76 Sample Debug Frame-Relay LAPF Output—LAPF State

```
1w0d# debug frame-relay lapf
1w0d: *LAPF Serial0 T203 expired, state = 7
1w0d: *LAPF Serial0 lapf_rr_7
1w0d: *LAPF Serial0 TX -> RR Rsp p/f=1, N(R)= 3
1w0d: *LAPF Serial0 RX <- RR Cmd p/f=1, N(R)= 3
1w0d: *LAPF Serial0 lapf_rr_7
1w0d: *LAPF Serial0 TX -> RR Rsp p/f=1, N(R)= 3
1w0d: *LAPF Serial0 RX <- RR Cmd p/f=1, N(R)= 3
1w0d: *LAPF Serial0 lapf_rr_7
```

debug frame-relay lmi

Use the **debug frame-relay lmi** EXEC command to display information on the local management interface (LMI) packets exchanged by the router and the Frame Relay service provider. The **no** form of this command disables debugging output.

[no] debug frame-relay lmi [interface name]

Syntax Description

interface name (Optional) Name of interface.

Usage Guidelines

You can use this command to determine whether the router and the Frame Relay switch are sending and receiving LMI packets properly.

Note Because the **debug frame-relay lmi** command does not generate much output, you can use it at any time, even during periods of heavy traffic, without adversely affecting other users on the system.

Sample Display

Figure 2-77 shows sample **debug frame-relay lmi** output.

Figure 2-77 Sample Debug Frame-Relay LMI Output

```
router# debug frame-relay lmi
```

LMI exchange	<pre>Serial1(out): StEnq, clock 20212760, myseq 206, mineseen 205, yourseen 136, DTE up Serial1(in): Status, clock 20212764, myseq 206 RT IE 1, length 1, type 1 KA IE 3, length 2, yourseq 138, myseq 206</pre>
Full LMI status message	<pre>Serial1(out): StEnq, clock 20222760, myseq 207, mineseen 206, yourseen 138, DTE up Serial1(in): Status, clock 20222764, myseq 207 RT IE 1, length 1, type 1 KA IE 3, length 2, yourseq 140, myseq 207 Serial1(out): clock 20232760, myseq 208, mineseen 207, yourseen 140, line up RT IE 1, length 1, type 1 KA IE 3, length 2, yourseq 142, myseq 208</pre>
	<pre>Serial1(out): StEnq, clock 20252760, myseq 210, mineseen 209, yourseen 144, DTE up Serial1(in): Status, clock 20252764, RT IE 1, length 1, type 0 KA IE 3, length 2, yourseq 146, myseq 210 PVC IE 0x7, length 0x6, dlci 400, status 0, bw 56000 PVC IE 0x7, length 0x6, dlci 401, status 0, bw 56000</pre>

S2546

In Figure 2-77, the first four lines describe an LMI exchange. The first line describes the LMI request the router has sent to the switch. The second line describes the LMI reply the router has received from the switch. The third and fourth lines describe the response to this request from the switch. This

LMI exchange is followed by two similar LMI exchanges. The last six lines in Figure 2-77 consist of a full LMI status message that includes a description of the router's two permanent virtual circuits (PVCs).

Table 2-38 describes significant fields in the first line of the **debug frame-relay lmi** output shown in Figure 2-77.

Table 2-38 Debug Frame-Relay LMI Field Descriptions—Part 1

Field	Description
Serial1(out)	Indication that the LMI request was sent out on the Serial1 interface.
StEnq	Command mode of message: StEnq—Status inquiry Status—Status reply
clock 20212760	System clock (in milliseconds). Useful for determining whether an appropriate amount of time has transpired between events.
myseq 206	The myseq counter maps to the router's CURRENT SEQ counter.
yourseen 136	The yourseen counter maps to the LAST RCVD SEQ counter of the switch.
DTE up	Line protocol up/down state for the DTE (user) port.

Table 2-39 describes significant fields in the third and fourth lines of **debug frame-relay lmi** output shown in Figure 2-77.

Table 2-39 Debug Frame-Relay LMI Field Descriptions—Part 2

Field	Description
RT IE 1	Value of the report type information element.
length 1	Length of the report type information element (in bytes).
type 1	Report type in RT IE.
KA IE 3	Value of the keepalive information element.
length 2	Length of the keepalive information element (in bytes).
yourseq 138	The yourseq counter maps to the CURRENT SEQ counter of the switch.
myseq 206	The myseq counter maps to the router's CURRENT SEQ counter.

Table 2-40 describes significant fields in the last line of **debug frame-relay lmi** output shown in Figure 2-77.

Table 2-40 Debug Frame-Relay LMI Field Descriptions—Part 3

Field	Description
PVC IE 0x7	Value of the permanent virtual circuit information element type.
length 0x6	Length of the PVC IE (in bytes).
dldci 401	DLCI decimal value for this PVC.

Table 2-40 Debug Frame-Relay LMI Field Descriptions—Part 3

Field	Description
status 0	Status value. Possible values include the following: 0x00—Added/inactive 0x02—Added/active 0x04—Deleted 0x08—New/inactive 0x0a—New/active
bw 56000	CIR (committed information rate), in decimal, for the DLCI.

debug frame-relay networklayerinterface

Use the **debug frame-relay networklayerinterface** EXEC command to display Network Layer Interface (NLI) information. The **no** form of this command disables debugging output.

[no] debug frame-relay networklayerinterface

Usage Guidelines

The Frame Relay SVC signaling subsystem is decoupled from the rest of the router code by means of the Network Layer Interface intermediate software layer.

The **debug frame-relay networklayerinterface** command shows what happens within the network layer interface when a call is set up or torn down. All output that contains an *NL* relate to the interaction between the Q.933 signaling subsystem and the Network Layer Interface.

Note The **debug frame-relay networklayerinterface** command has no significance to anyone who is not familiar with the inner workings of the Cisco IOS software. This command is typically used by service personnel to debug problem situations.

Sample Displays

Figure 2-78 shows sample **debug frame-relay networklayerinterface** output. This example displays the output generated when a call is set up. Figure 2-79 shows an example of the output generated when a call is torn down.

Figure 2-78 Sample Debug Frame-Relay Network Layer Interface Output—Call Setup

```
1w0d# debug frame-relay networklayerinterface
1w0d: NLI STATE: L3_CALL_REQ, Call ID 1 state 0
1w0d: NLI: Walking the event table 1
1w0d: NLI: Walking the event table 2
1w0d: NLI: Walking the event table 3
1w0d: NLI: Walking the event table 4
1w0d: NLI: Walking the event table 5
1w0d: NLI: Walking the event table 6
1w0d: NLI: Walking the event table 7
1w0d: NLI: Walking the event table 8
1w0d: NLI: Walking the event table 9
1w0d: NLI: NL0_L3CallReq
1w0d: NLI: State: STATE_NL_NULL, Event: L3_CALL_REQ, Next: STATE_L3_CALL_REQ
1w0d: NLI: Enqueued outgoing packet on holdq
1w0d: NLI: Map-list search: Found maplist bermuda
1w0d: daddr.subaddr 0, saddr.subaddr 0, saddr.subaddr 0
1w0d: saddr.subaddr 0, daddr.subaddr 0, daddr.subaddr 0
1w0d: nli_parameter_negotiation
1w0d: NLI STATE: NL_CALL_CNF, Call ID 1 state 10
1w0d: NLI: Walking the event table 1
1w0d: NLI: Walking the event table 2
1w0d: NLI: Walking the event table 3
1w0d: NLI: NLx_CallCnf
1w0d: NLI: State: STATE_L3_CALL_REQ, Event: NL_CALL_CNF, Next: STATE_NL_CALL_CNF
1w0d: Checking maplist "junk"
1w0d: working with maplist "bermuda"
1w0d: Checking maplist "bermuda"
1w0d: working with maplist "bermuda"
1w0d: NLI: Emptying holdQ, link 7, dlci 100, size 104
```

Figure 2-79 Sample Debug Frame-Relay Network Layer Interface Output—Call Teardown

```

1w0d# debug frame-relay networklayerinterface
1w0d: NLI: L3 Call Release Req for Call ID 1
1w0d: NLI STATE: L3_CALL_REL_REQ, Call ID 1 state 3
1w0d: NLI: Walking the event table 1
1w0d: NLI: Walking the event table 2
1w0d: NLI: Walking the event table 3
1w0d: NLI: Walking the event table 4
1w0d: NLI: Walking the event table 5
1w0d: NLI: Walking the event table 6
1w0d: NLI: Walking the event table 7
1w0d: NLI: Walking the event table 8
1w0d: NLI: Walking the event table 9
1w0d: NLI: Walking the event table 10
1w0d: NLI: NLx_L3CallRej
1w0d: NLI: State: STATE_NL_CALL_CNF, Event: L3_CALL_REL_REQ, Next: STATE_L3_CALL_REL_REQ
1w0d: NLI: junk: State: STATE_NL_NULL, Event: L3_CALL_REL_REQ, Next: STATE_NL_NULL
1w0d: NLI: Map-list search: Found maplist junk
1w0d: daddr.subaddr 0, saddr.subaddr 0, saddr.subaddr 0
1w0d: saddr.subaddr 0, daddr.subaddr 0, daddr.subaddr 0
1w0d: nli_parameter_negotiation
1w0d: NLI STATE: NL_REL_CNF, Call ID 1 state 0
1w0d: NLI: Walking the event table 1
1w0d: NLI: Walking the event table 2
1w0d: NLI: Walking the event table 3
1w0d: NLI: Walking the event table 4
1w0d: NLI: Walking the event table 5
1w0d: NLI: Walking the event table 6
1w0d: NLI: Walking the event table 7
1w0d: NLI: NLx_RelCnf
1w0d: NLI: State: STATE_NL_NULL, Event: NL_REL_CNF, Next: STATE_NL_NULL
    
```

Table 2-41 describes the states and events shown in Figure 2-78 and Figure 2-79.

Table 2-41 Network Layer Interface State and Event Descriptions

State and Event	Description
L3_CALL_REQ	Internal call setup request. Network layer indicates that a switched virtual circuit (SVC) is required.
STATE_NL_NULL	Call in initial state—no call exists.
STATE_L3_CALL_REQ	Setup message sent out and waiting for a reply. This is the state the network layer state machine transitions to when a call request is received from Layer 3 but no confirmation has been received from the network.
NL_CALL_CNF	Message sent from Q.933 signaling subsystem to the Network Layer Interface asking that internal resources be allocated for the call.
STATE_L3_CALL_CNF	Q.933 state indicating that the call is active. After the network confirms a call request using a connect message, the Q.933 state machine transitions to this state.
STATE_NL_CALL_CNF	Internal software state indicating software resources are assigned and the call is up. After Q.933 transitions to the STATE_L3_CALL_CNF state, it sends an NL_CALL_CNF message to the network layer state machine, which then transitions to the STATE_NL_CALL_CNF state.
L3_CALL_REL_REQ	Internal request to release the call.

Table 2-41 Network Layer Interface State and Event Descriptions (Continued)

State and Event	Description
STATE_L3_CALL_REL_REQ	Internal software state indicating the call is in the process of being released. At this point, the Q.933 subsystem is told that the call is being released and a disconnect message goes out for the Q.933 subsystem.
NL_REL_CNF	Indication from the Q.933 signaling subsystem that the signaling subsystem is releasing the call. After receiving a release complete message from the network indicating that the release process is complete, the Q.933 subsystem sends an NL_REL_CNF event to the network layer subsystem.

Related Command**debug frame-relay callcontrol**

debug frame-relay packet

Use the **debug frame-relay packet** EXEC command to display information on packets that have been sent on a Frame Relay interface. The **no** form of this command disables debugging output.

[no] debug frame-relay packet [interface name [dlci value]]

Syntax Description

- interface name** (Optional) Name of interface or subinterface.
- dlci value** (Optional) Data link connection identifier (DLCI) decimal value.

Usage Guidelines

This command helps you analyze the packets that are sent on a Frame Relay interface. Because the **debug frame-relay packet** command generates large amounts of output, only use it when traffic on the Frame Relay network is less than 25 packets per second. Use the options to limit the debugging output to a specific DLCI or interface.

To analyze the packets *received* on a Frame Relay interface, use the **debug frame-relay** command.

Sample Display

Figure 2-80 shows sample **debug frame-relay packet** output.

Figure 2-80 Sample Debug Frame-Relay Packet Output

```
router# debug frame-relay packets
```

```
Serial0: broadcast = 1, link 809B, addr 65535.255
Serial0(o):DLCI 500 type 809B size 24
```

Groups of
output lines

```
Serial0: broadcast - 0, link 809B, addr 10.2
Serial0(o):DLCI 100 type 809B size 104
Serial0: broadcast search
Serial0(o):DLCI 300 type 809B size 24
Serial0(o):DLCI 400 type 809B size 24
```

S2547

As Figure 2-80 shows, **debug frame-relay packet** output consists of groups of output lines; each group describes a Frame Relay packet that has been sent. The number of lines in the group can vary, depending on the number of data link connection identifiers (DLCIs) on which the packet was sent. For example, the first two pairs of output lines describe two different packets, both of which were sent out on a single DLCI. The last three lines in Figure 2-80 describe a single Frame Relay packet that was sent out on two DLCIs.

Table 2-42 describes significant fields shown in the first pair of output lines in Figure 2-80.

Table 2-42 Debug Frame-Relay Packet Field Descriptions

Field	Description
Serial0:	Interface that has sent the Frame Relay packet.

Table 2-42 Debug Frame-Relay Packet Field Descriptions (Continued)

Field	Description
broadcast = 1	Destination of the packet. Possible values include the following: broadcast = 1—Broadcast address broadcast = 0—Particular destination broadcast search—Searches all Frame Relay map entries for this particular protocol that include the keyword broadcast .
link 809B	Link type, as documented under debug frame-relay .
addr 65535.255	Destination protocol address for this packet. In this case, it is an AppleTalk address.
Serial0(o):	(o) indicates that this is an output event.
DLCI 500	Decimal value of the DLCI.
type 809B	Packet type, as documented under debug frame-relay .
size 24	Size of this packet (in bytes).

Explanations for other lines of output shown in Figure 2-80 follow:

The following lines describe a Frame Relay packet sent to a particular address; in this case AppleTalk address 10.2:

```
Serial0: broadcast - 0, link 809B, addr 10.2
Serial0(o):DLCI 100 type 809B size 104
```

The following lines describe a Frame Relay packet that went out on two different DLCIs, because two Frame Relay map entries were found:

```
Serial0: broadcast search
Serial0(o):DLCI 300 type 809B size 24
Serial0(o):DLCI 400 type 809B size 24
```

The following lines do not appear in Figure 2-80. They describe a Frame Relay packet sent to a true broadcast address.

```
Serial1: broadcast search
Serial1(o):DLCI 400 type 800 size 288
```

debug fras error

Use the **debug fras error** EXEC command to display information about Frame Relay Access Support (FRAS) protocol errors. The **no** form of this command disables debugging output.

[no] debug fras error

Usage Guidelines

For complete information on the FRAS process, use the **debug fras message** along with the **debug fras error** command.

Sample Display

Figure 2-81 shows sample **debug fras error** output. This example shows that no logical connection exists between the local station and remote station in the current setup.

Figure 2-81 Sample Debug FRAS Error Output

```
router# debug fras error

FRAS: No route, lmac 1000.5acc.7fb1 rmac 4fff.0000.0000, lSap=0x4, rSap=0x4
FRAS: Can not find the Setup
```

Related Commands

debug cls message
debug fras message
debug fras state

debug fras message

Use the **debug fras message** EXEC command to display general information about Frame Relay Access Support (FRAS) messages. The **no** form of this command disables debugging output.

[no] debug fras message

Usage Guidelines

For complete information on the FRAS process, use the **debug fras error** along with the **debug fras message** command.

Sample Display

Figure 2-82 shows sample **debug fras message** output. This example shows incoming Cisco Link Services (CLS) primitives.

Figure 2-82 Sample Debug FRAS Message Output

```
router# debug fras message

FRAS: receive 4C23
FRAS: receive CC09
```

Related Commands

debug cls message
debug fras error
debug fras state

debug fras state

Use the **debug fras state** EXEC command to display information about Frame Relay Access Support (FRAS) data link control link state changes. The **no** form of this command disables debugging output.

[no] debug fras state

Sample Display

Figure 2-83 shows sample **debug fras state** output. This example shows the state changing from a *request open station is sent* state to an *exchange XID* state.

Possible states are the following: reset, request open station is sent, exchange xid, connection request is sent, signal station wait, connection response wait, connection response sent, connection established, disconnect wait, and number of link states.

Figure 2-83 Sample Debug FRAS State Output

```
router# debug fras state

FRAS: TR0 (04/04) oldstate=LS_RQOPNSTNSENT, input=RQ_OPNSTN_CNF
FRAS: newstate=LS_EXCHGXID
```

Related Commands

debug cls message
debug fras error
debug fras message

debug ip drp

Use the **debug ip drp** EXEC command to display Director Response Protocol (DRP) information. The **no** form of this command disables debugging output.

[no] debug ip drp

Usage Guidelines

The **debug ip drp** command is used to debug the director response agent used by the Distributed Director product. The Distributed Director can be used to dynamically respond to Domain Name System (DNS) queries with the IP address of the “best” host based on various criteria.

Sample Display

Figure 2-84 shows sample **debug ip drp** output. This example shows the packet origination, the IP address that information is routed to, and the route metrics that were returned.

Figure 2-84 Sample Debug IP DRP Output

```
1d02hr# debug ip drp
1d02h: DRP: received v1 packet from 172.31.232.8, via Ethernet0
1d02h: DRP: RTQUERY for 172.31.58.94 returned internal=0,external=0
```

Table 2-43 describes the fields shown in Figure 2-84.

Table 2-43 Debug IP DRP Field Descriptions

Field	Description
v1 packet	Version 1 packet.
internal	If nonzero, the metric for the internal distance of the route that the router uses to send packets in the direction of the client. The internal distance is the distance within the router’s autonomous system.
external	If nonzero, the metric for the Border Gateway Protocol (BGP) or external distance used to send packets to the client. The external distance is the distance outside the router’s autonomous system.

debug ip dvmrp

Use the **debug ip dvmrp** EXEC command to display information on Distance Vector Multiprotocol Routing Protocol (DVMRP) packets received and transmitted. The **no** form of this command disables debugging output.

```
[no] debug ip dvmrp [detail [access-list] [in | out]]
```

Syntax Description

detail	(Optional) Enables a more detailed level of output and displays packet contents.
<i>access-list</i>	(Optional) Causes the debug ip dvmrp command to restrict output to one access list.
in	(Optional) Causes the debug ip dvmrp command to output packets received in DVMRP reports.
out	(Optional) Causes the debug ip dvmrp command to output packets transmitted in DVMRP reports.

Usage Guidelines

Use the **debug ip dvmrp detail** command with care. This command generates a great deal of output and can interrupt other activity on the router when it is invoked.

Sample Display

Figure 2-85 shows sample **debug ip dvmrp** output.

Figure 2-85 Sample Debug IP DVMRP Output

```
router# debug ip dvmrp
DVMRP: Received Report on Ethernet0 from 172.19.244.10
DVMRP: Received Report on Ethernet0 from 172.19.244.11
DVMRP: Building Report for Ethernet0 224.0.0.4
DVMRP: Send Report on Ethernet0 to 224.0.0.4
DVMRP: Sending IGMP Reports for known groups on Ethernet0
DVMRP: Received Report on Ethernet0 from 172.19.244.10
DVMRP: Received Report on Tunnel0 from 192.168.199.254
DVMRP: Received Report on Tunnel0 from 192.168.199.254
DVMRP: Received Report on Tunnel0 from 192.168.199.254
DVMRP: Received Report on Tunnel0 from 192.168.199.254
DVMRP: Received Report on Tunnel0 from 192.168.199.254
DVMRP: Received Report on Tunnel0 from 192.168.199.254
DVMRP: Building Report for Tunnel0 224.0.0.4
DVMRP: Send Report on Tunnel0 to 192.168.199.254
DVMRP: Send Report on Tunnel0 to 192.168.199.254
DVMRP: Send Report on Tunnel0 to 192.168.199.254
DVMRP: Send Report on Tunnel0 to 192.168.199.254
DVMRP: Radix tree walk suspension
DVMRP: Send Report on Tunnel0 to 192.168.199.254
```

Explanations for individual lines of output from Figure 2-85 follow.

The following lines show that the router received DVMRP routing information and placed it in the mroute table:

```
DVMRP: Received Report on Ethernet0 from 172.19.244.10
DVMRP: Received Report on Ethernet0 from 172.19.244.11
```

The following lines show that the router is creating a report to send to another DVMRP router:

```
DVMRP: Building Report for Ethernet0 224.0.0.4
DVMRP: Send Report on Ethernet0 to 224.0.0.4
```

Table 2-44 provides a list of internet multicast addresses supported for host IP implementations.

Table 2-44 Internet Multicast Addresses

Address	Description	RFC
224.0.0.0	Base address (Reserved)	RFC 1112
224.0.0.1	All systems on this subnet	RFC 1112
224.0.0.2	All routers on this subnet	
224.0.0.3	Unassigned	
224.0.0.4	DVMRP routers	RFC 1075
224.0.0.5	OSPF/IGP all routers	RFC 1583

The following lines show that a protocol update report has been sent to all known multicast groups. Hosts use IGMP reports to communicate with routers and to request to join a multicast group. In this case, the router is sending an IGMP report for every known group to the host, which is running mroute. The host then responds as though the router was a host on the LAN segment that wants to receive multicast packets for the group.

```
DVMRP: Sending IGMP Reports for known groups on Ethernet0
```

Figure 2-86 shows sample **debug ip dvmrp detail** output.

Figure 2-86 Sample Debug IP DVMRP Detail Output

```
router# debug ip dvmrp detail

DVMRP: Sending IGMP Reports for known groups on Ethernet0
DVMRP: Advertise group 224.2.224.2 on Ethernet0
DVMRP: Advertise group 224.2.193.34 on Ethernet0
DVMRP: Advertise group 224.2.231.6 on Ethernet0
DVMRP: Received Report on Tunnel0 from 192.168.199.254
DVMRP: Origin 150.166.53.0/24, metric 13, distance 0
DVMRP: Origin 150.166.54.0/24, metric 13, distance 0
DVMRP: Origin 150.166.55.0/24, metric 13, distance 0
DVMRP: Origin 150.166.56.0/24, metric 13, distance 0
DVMRP: Origin 150.166.92.0/24, metric 12, distance 0
DVMRP: Origin 150.166.100.0/24, metric 12, distance 0
DVMRP: Origin 150.166.101.0/24, metric 12, distance 0
DVMRP: Origin 150.166.142.0/24, metric 8, distance 0
DVMRP: Origin 150.166.200.0/24, metric 12, distance 0
DVMRP: Origin 150.166.237.0/24, metric 12, distance 0
DVMRP: Origin 150.203.5.0/24, metric 8, distance 0
```

Explanations for individual lines of output from Figure 2-86 follow.

The following lines show that this group is available to the DVMRP router. The mrouterd process on the host will forward the source and multicast information for this group through the DVMRP cloud to other members.

```
DVMRP: Advertise group 224.2.224.2 on Ethernet0
```

The following lines show the DVMRP route information:

```
DVMRP: Origin 150.166.53.0/24, metric 13, distance 0  
DVMRP: Origin 150.166.54.0/24, metric 13, distance 0
```

Metric is the number of hops the route has covered. Distance is the administrative distance.

debug ip eigrp

Use the **debug ip eigrp** EXEC command to display information on Enhanced IGRP protocol packets. The **no** form of this command disables debugging output.

[no] debug ip eigrp

Usage Guidelines

This command helps you analyze the packets that are sent and received on an interface. Because the **debug ip eigrp** command generates large amounts of output, only use it when traffic on the network is light.

Sample Display

Figure 2-87 shows sample **debug ip eigrp** output.

Figure 2-87 Sample Debug IP EIGRP Output

```
router# debug ip eigrp

IP-EIGRP: Processing incoming UPDATE packet
IP-EIGRP: Ext 192.168.3.0 255.255.255.0 M 386560 - 256000 130560 SM 360960 - 256
000 104960
IP-EIGRP: Ext 192.168.0.0 255.255.255.0 M 386560 - 256000 130560 SM 360960 - 256
000 104960
IP-EIGRP: Ext 192.168.3.0 255.255.255.0 M 386560 - 256000 130560 SM 360960 - 256
000 104960
IP-EIGRP: 172.24.43.0 255.255.255.0, - do advertise out Ethernet0/1
IP-EIGRP: Ext 172.24.43.0 255.255.255.0 metric 371200 - 256000 115200
IP-EIGRP: 192.135.246.0 255.255.255.0, - do advertise out Ethernet0/1
IP-EIGRP: Ext 192.135.246.0 255.255.255.0 metric 46310656 - 45714176 596480
IP-EIGRP: 172.24.40.0 255.255.255.0, - do advertise out Ethernet0/1
IP-EIGRP: Ext 172.24.40.0 255.255.255.0 metric 2272256 - 1657856 614400
IP-EIGRP: 192.135.245.0 255.255.255.0, - do advertise out Ethernet0/1
IP-EIGRP: Ext 192.135.245.0 255.255.255.0 metric 40622080 - 40000000 622080
IP-EIGRP: 192.135.244.0 255.255.255.0, - do advertise out Ethernet0/1
```

Table 2-45 describes significant fields in the debug messages shown in Figure 2-87.

Table 2-45 Debug IP EIGRP Field Descriptions

Field	Description
IP-EIGRP:	Indicates that this is an IP Enhanced IGRP packet.
Ext	Indicates the following address is an external destination rather than an internal destination, which would be labeled as Int.
M	Shows the computed metric, which includes SM and the cost between this router and the neighbor. The first number is the composite metric. The next two numbers are the inverse bandwidth and the delay, respectively.
SM	Shows the metric as reported by the neighbor.

debug ip http ezsetup

Use the **debug ip http ezsetup** EXEC command to display the configuration changes that occur during the EZ Setup process. The **no** form of this command disables debugging output.

[no] debug ip http ezsetup

Usage Guidelines

Use the **debug ip http ezsetup** command to verify the EZ Setup actions without changing the router's configuration.

EZ Setup is a form you fill out to perform basic router configuration from most Hypertext Markup Language (HTML) browsers.

Sample Display

Figure 2-88 shows sample **debug ip http ezsetup** output. This example shows the configuration changes for the router when the EZ Setup form has been submitted.

Figure 2-88 Sample Debug IP HTTP EZ Setup Output

```

router#>debug ip http ezsetup
service timestamps debug
service timestamps log
service password-encryption
!
hostname router-name
!
enable secret router-pw
line vty 0 4
password router-pw
!
interface ethernet 0
 ip address 172.21.52.9 255.255.255.0
 no shutdown
 ip helper-address 172.31.2.132
 ip name-server 172.31.2.132
 isdn switch-type basic-5ess
 username Remote-name password Remote-chap
interface bri 0
 ip unnumbered ethernet 0
 encapsulation ppp
 no shutdown
 dialer map ip 192.168.254.254 speed 56 name Remote-name Remote-number
 isdn spid1 spid1
 isdn spid2 spid2
 ppp authentication chap callin
 dialer-group 1
!
ip classless
access-list 101 deny udp any any eq snmp
access-list 101 deny udp any any eq ntp
access-list 101 permit ip any any
dialer-list 1 list 101
ip route 0.0.0.0 0.0.0.0 192.168.254.254
ip route 192.168.254.254 255.255.255.255 bri 0
logging buffered
snmp-server community public RO
ip http server

```

```
ip classless
ip subnet-zero
!
end
```

Related Commands

debug ip http token
debug ip http transaction
debug ip http url

debug ip http token

Use the **debug ip http token** EXEC command to display individual tokens parsed by the Hypertext Transfer Protocol (HTTP) server. The **no** form of this command disables debugging output.

[no] debug ip http token

Usage Guidelines

Use the **debug ip http token** command to display what the HTTP server is parsing at a low level. To display what the HTTP server is parsing at a high level, use the **debug ip http transaction** command.

Sample Display

Figure 2-89 shows a partial sample of **debug ip http token** output. In this example, the browser accessed the router's home page *http://router-name/*. The output gives the token parsed by the HTTP server and its length.

Figure 2-89 Sample Debug IP HTTP Token Output

```
router#>debug ip http token
HTTP: token len 3: 'GET'
HTTP: token len 1: ' '
HTTP: token len 1: '/'
HTTP: token len 1: ' '
HTTP: token len 4: 'HTTP'
HTTP: token len 1: '/'
HTTP: token len 1: '1'
HTTP: token len 1: '.'
HTTP: token len 1: '0'
HTTP: token len 2: '\15\12'
HTTP: token len 7: 'Referer'
HTTP: token len 1: ':'
HTTP: token len 1: ' '
HTTP: token len 4: 'http'
HTTP: token len 1: ':'
HTTP: token len 1: '/'
HTTP: token len 1: '/'
HTTP: token len 3: 'www'
HTTP: token len 1: '.'
HTTP: token len 3: 'foo'
HTTP: token len 1: '.'
HTTP: token len 3: 'com'
HTTP: token len 1: '/'
HTTP: token len 2: '\15\12'
HTTP: token len 10: 'Connection'
HTTP: token len 1: ':'
HTTP: token len 1: ' '
HTTP: token len 4: 'Keep'
HTTP: token len 1: '-'
HTTP: token len 5: 'Alive'
HTTP: token len 2: '\15\12'
HTTP: token len 4: 'User'
HTTP: token len 1: '-'
HTTP: token len 5: 'Agent'
HTTP: token len 1: ':'
HTTP: token len 1: ' '
HTTP: token len 7: 'Mozilla'
HTTP: token len 1: '/'
```

```
HTTP: token len 1: '2'  
HTTP: token len 1: '.'  
...
```

Related Commands

debug ip http ezsetup

debug ip http transaction

debug ip http url

debug ip http transaction

Use the **debug ip http transaction** EXEC command to display Hypertext Transfer Protocol (HTTP) server transaction processing. The **no** form of this command disables debugging output.

[no] debug ip http transaction

Usage Guidelines

Use the **debug ip http transaction** command to display what the HTTP server is parsing at a high level. To display what the HTTP server is parsing at a low level, use the **debug ip http token** command.

Sample Display

Figure 2-90 shows sample **debug ip http transaction** output. In this example, the browser accessed the router's home page *http://router-name/*.

Figure 2-90 Sample Debug IP HTTP Transaction Output

```
router#>debug ip http transaction
HTTP: parsed uri '/'
HTTP: client version 1.0
HTTP: parsed extension Referer
HTTP: parsed line http://www.foo.com/
HTTP: parsed extension Connection
HTTP: parsed line Keep-Alive
HTTP: parsed extension User-Agent
HTTP: parsed line Mozilla/2.01 (X11; I; FreeBSD 2.1.0-RELEASE i386)
HTTP: parsed extension Host
HTTP: parsed line router-name
HTTP: parsed extension Accept
HTTP: parsed line image/gif, image/x-xbitmap, image/jpeg, image/
HTTP: parsed extension Authorization
HTTP: parsed authorization type Basic
HTTP: received GET ''
```

Table 2-46 lists describes some of the fields in the example shown above.

Table 2-46 Debut IP HTTP Transaction Field Descriptions

Field	Description
HTTP: parsed uri '/'	Uniform resource identifier that is requested.
HTTP: client version 1.0	Client HTTP version.
HTTP: parsed extension Referer	HTTP extension.
HTTP: parsed line http://www.foo.com/	Value of HTTP extension.
HTTP: received GET ''	HTTP request method.

Related Commands

debug ip http ezsetup
debug ip http token
debug ip http url

debug ip http url

Use the **debug ip http url** EXEC command to show the uniform resource locators (URLs) accessed from the router. The **no** form of this command disables debugging output.

[no] debug ip http url

Usage Guidelines

Use the **debug ip http url** command to keep track of the URLs that are accessed and to determine from which hosts the URLs are accessed.

Sample Display

Figure 2-91 shows a partial sample of **debug ip http url** output. In this example, the HTTP server accessed the URLs */* and */exec*. The output shows the URL being requested and the IP address of the host requesting the URL.

Figure 2-91 Sample Debug IP HTTP URL Output

```
router#>debug ip http url
HTTP: processing URL '/' from host 172.31.2.141
HTTP: processing URL '/exec' from host 172.31.2.141
```

Related Commands

debug ip http ezsetup

debug ip http token

debug ip http transaction

debug ip icmp

Use the **debug ip icmp** EXEC command to display information on Internet Control Message Protocol (ICMP) transactions. The **no** form of this command disables debugging output.

[no] debug ip icmp

Usage Guidelines

This command helps you determine whether the router is sending or receiving ICMP messages. Use it, for example, when you are troubleshooting an end-to-end connection problem.

Note For more information about the fields in **debug ip icmp** output, see RFC-792, “Internet Control Message Protocol”; Appendix I of RFC-950, “Internet Standard Subnetting Procedure”; and RFC-1256, “ICMP Router Discovery Messages.”

Sample Display

Figure 2-92 shows sample **debug ip icmp** output.

Figure 2-92 Sample Debug IP ICMP Output

```
router# debug ip icmp

ICMP: rcvd type 3, code 1, from 10.95.192.4
ICMP: src 10.56.0.202, dst 172.16.16.1, echo reply
ICMP: dst (10.120.1.0) port unreachable rcv from 10.120.1.15
ICMP: src 172.16.12.35, dst 172.16.20.7, echo reply
ICMP: dst (255.255.255.255) protocol unreachable rcv from 10.31.7.21
ICMP: dst (10.120.1.0) port unreachable rcv from 10.120.1.15
ICMP: dst (255.255.255.255) protocol unreachable rcv from 10.31.7.21
ICMP: dst (10.120.1.0) port unreachable rcv from 10.120.1.15
ICMP: src 10.56.0.202, dst 172.16.16.1, echo reply
ICMP: dst (10.120.1.0) port unreachable rcv from 10.120.1.15
ICMP: dst (255.255.255.255) protocol unreachable rcv from 10.31.7.21
ICMP: dst (10.120.1.0) port unreachable rcv from 10.120.1.15
```

Table 2-47 describes significant fields in the first line of **debug ip icmp** output shown in Figure 2-92.

Table 2-47 Debug IP ICMP Field Descriptions—Part 1

Field	Description
ICMP:	Indication that this message describes an ICMP packet.
rcvd type 3	<p>The type field can be one of the following:</p> <ul style="list-style-type: none"> • 0—Echo Reply • 3—Destination Unreachable • 4—Source Quench • 5—Redirect • 8—Echo • 9—Router Discovery Protocol Advertisement • 10—Router Discovery Protocol Solicitations • 11—Time Exceeded • 12—Parameter Problem • 13—Timestamp • 14—Timestamp Reply • 15—Information Request • 16—Information Reply • 17—Mask Request • 18—Mask Reply
code 1	<p>This field is a code. The meaning of the code depends upon the type field value:</p> <ul style="list-style-type: none"> • Echo and Echo Reply—The code field is always zero. • Destination Unreachable—The code field can have the following values: <ul style="list-style-type: none"> — 0—Network unreachable — 1—Host unreachable — 2—Protocol unreachable — 3—Port unreachable — 4—Fragmentation needed and DF bit set — 5—Source route failed • Source Quench—The code field is always 0. • Redirect—The code field can have the following values: <ul style="list-style-type: none"> — 0—Redirect datagrams for the network — 1—Redirect datagrams for the host — 2—Redirect datagrams for the command mode of service and network — 3—Redirect datagrams for the command mode of service and host • Router Discovery Protocol Advertisements and Solicitations—The code field is always zero.

Table 2-47 Debug IP ICMP Field Descriptions—Part 1 (Continued)

Field	Description
code 1 (continued)	<ul style="list-style-type: none"> • Time Exceeded—The code field can have the following values: <ul style="list-style-type: none"> — 0—Time to live exceeded in transit — 1—Fragment reassembly time exceeded • Parameter Problem—The code field can have the following values: <ul style="list-style-type: none"> — 0—General problem — 1—Option is missing — 2—Option missing, no room to add • Timestamp and Timestamp Reply—The code field is always zero. • Information Request and Information Reply—The code field is always zero. • Mask Request and Mask Reply—The code field is always zero.
from 10.95.192.4	Source address of the ICMP packet.

Table 2-48 describes significant fields in the second line of **debug ip icmp** output in Figure 2-92.

Table 2-48 Debug IP ICMP Field Descriptions—Part 2

Field	Description
ICMP:	Indication that this message describes an ICMP packet
src 10.5610.120.0.202	The address of the sender of the echo
dst 172.16.16.1	The address of the receiving router
echo reply	Indication the router received an echo reply

Other messages that the **debug ip icmp** command can generate follow.

When an IP router or host sends out an ICMP mask request, the following message is generated when the router sends a mask reply:

```
ICMP: sending mask reply (255.255.255.0) to 172.21.80.23 via Ethernet0
```

The following two lines are examples of the two forms of this message. The first form is generated when a mask reply comes in after the router sends out a mask request. The second form occurs when the router receives a mask reply with a nonmatching sequence and ID. See Appendix I of RFC 950, “Internet Standard Subnetting Procedures,” for details.

```
ICMP: mask reply 255.255.255.0 from 172.21.80.31
ICMP: unexpected mask reply 255.255.255.0 from 172.21.80.32
```

The following output indicates that the router sent a redirect packet to the host at address 172.21.80.31, instructing that host to use the gateway at address 172.21.80.23 in order to reach the host at destination address 172.16.1.111:

```
ICMP: redirect sent to 172.21.80.31 for dest 172.16.1.111 use gw 172.21.80.23
```

The following message indicates that the router received a redirect packet from the host at address 172.21.80.23, instructing the router to use the gateway at address 172.21.80.28 in order to reach the host at destination address 172.21.81.34:

```
ICMP: redirect rcvd from 172.21.80.23 -- for 172.21.81.34 use gw 172.21.80.28
```

The following message is displayed when the router sends an ICMP packet to the source address (172.21.94.31 in this case), indicating that the destination address (172.16.13.33 in this case) is unreachable:

```
ICMP: dst (172.16.13.33) host unreachable sent to 172.21.94.31
```

The following message is displayed when the router receives an ICMP packet from an intermediate address (172.21.98.32 in this case), indicating that the destination address (172.16.13.33 in this case) is unreachable:

```
ICMP: dst (172.16.13.33) host unreachable rcv from 172.21.98.32
```

Depending on the code received (as Table 2-47 describes), any of the unreachable messages can have any of the following “strings” instead of the “host” string in the message:

```
net
protocol
port
frag. needed and DF set
source route failed
prohibited
```

The following message is displayed when the TTL in the IP header reaches zero and a time exceed ICMP message is sent. The fields are self-explanatory.

```
ICMP: time exceeded (time to live) send to 10.95.1.4 (dest was 172.16.1.111)
```

The following message is generated when parameters in the IP header are corrupted in some way and the parameter problem ICMP message is sent. The fields are self-explanatory.

```
ICMP: parameter problem sent to 128.121.1.50 (dest was 172.16.1.111)
```

Based on the preceding information, the remaining output can be easily understood.

```
ICMP: parameter problem rcvd 172.21.80.32
ICMP: source quench rcvd 172.21.80.32
ICMP: source quench sent to 128.121.1.50 (dest was 172.16.1.111)
ICMP: sending time stamp reply to 172.21.80.45
ICMP: sending info reply to 172.21.80.12
ICMP: rdp advert rcvd type 9, code 0, from 172.21.80.23
ICMP: rdp solicit rcvd type 10, code 0, from 172.21.80.43
```

debug ip igmp

Use the **debug ip igmp** EXEC command to display Internet Group Management Protocol (IGMP) packets received and transmitted, as well as IGMP-host related events. The **no** form of this command disables debugging output.

[no] debug ip igmp

Usage Guidelines

This command helps discover whether the IGMP processes are functioning. In general, if IGMP is not working, the router process never discovers that there is another host on the network that is configured to receive multicast packets. In dense mode this means the packets will be delivered intermittently (a few every 3 minutes). In sparse mode they will never be delivered.

Use this command in conjunction with **debug ip pim** and **debug ip mrouting** to observe additional multicast activity and to see what is happening to the multicast routing process, or why packets are forwarded out of particular interfaces.

Sample Display

Figure 2-93 shows sample **debug ip igmp** output.

Figure 2-93 Sample Debug IP IGMP Output

```
router# debug ip igmp

IGMP: Received Host-Query from 172.24.37.33 (Ethernet1)
IGMP: Received Host-Report from 172.24.37.192 (Ethernet1) for 224.0.255.1
IGMP: Received Host-Report from 172.24.37.57 (Ethernet1) for 224.2.127.255
IGMP: Received Host-Report from 172.24.37.33 (Ethernet1) for 225.2.2.2
```

The messages displayed by the **debug ip igmp** command show query and report activity received from other routers and multicast group addresses.

Related Commands

debug ip pim

debug ip mrouting

debug ip igrp events

Use the **debug ip igrp events** EXEC command to display summary information on Interior Gateway Routing Protocol (IGRP) routing messages that indicates the source and destination of each update, as well as the number of routes in each update. Messages are not generated for each route. The **no** form of this command disables debugging output.

[no] debug ip igrp events [*ip-address*]

Syntax Description

ip-address (Optional) IP address of an IGRP neighbor.

Usage Guidelines

If the IP address of an IGRP neighbor is specified, the resulting **debug ip igrp events** output includes messages describing updates from that neighbor and updates that the router broadcasts toward that neighbor.

This command is particularly useful when there are many networks in your routing table. In this case, using **debug ip igrp transaction** could flood the console and make the router unusable. Use **debug ip igrp events** instead to display summary routing information.

Sample Display

Figure 2-94 shows sample **debug ip igrp events** output.

Figure 2-94 Sample Debug IP IGRP Events Output

```
router# debug ip igrp events
Updates sent to these two destination addresses — IGRP: sending update to 255.255.255.255 via Ethernet1 (160.89.33.8)
                                                    IGRP: Update contains 26 interior, 40 system, and 3 exterior routes.
                                                    IGRP: Total routes in update: 69
Updates received from these source addresses — IGRP: sending update to 255.255.255.255 via Ethernet0 (160.89.32.8)
                                                    IGRP: Update contains 1 interior, 0 system, and 0 exterior routes.
                                                    IGRP: Total routes in update: 1
                                                    IGRP: received update from 160.89.32.24 on Ethernet0
                                                    IGRP: Update contains 17 interior, 1 system, and 0 exterior routes.
                                                    IGRP: Total routes in update: 18
                                                    IGRP: received update from 160.89.32.7 on Ethernet0
                                                    IGRP: Update contains 5 interior, 1 system, and 0 exterior routes.
                                                    IGRP: Total routes in update: 6
```

S2548

Figure 2-94 shows that the router has sent two updates to the broadcast address 255.255.255.255. The router also received two updates. Three lines of output describe each of these updates.

The first line indicates whether the router sent or received the update packet, the source or destination address, and the interface through which the update was sent or received. If the update was sent, the IP address assigned to this interface is shown (in parentheses).

```
IGRP: sending update to 255.255.255.255 via Ethernet1 (160.89.33.8)
```

The second line summarizes the number and types of routes described in the update:

```
IGRP: Update contains 26 interior, 40 system, and 3 exterior routes.
```

The third line indicates the total number of routes described in the update.

```
IGRP: Total routes in update: 69
```

debug ip igrp transaction

Use the **debug ip igrp transaction** EXEC command to display transaction information on Interior Gateway Routing Protocol (IGRP) routing transactions. The **no** form of this command disables debugging output.

[no] debug ip igrp transaction [*ip-address*]

Syntax Description

ip-address (Optional) IP address of an IGRP neighbor.

Usage Guidelines

If the IP address of an IGRP neighbor is specified, the resulting **debug ip igrp transaction** output includes messages describing updates from that neighbor and updates that the router broadcasts toward that neighbor.

When there are many networks in your routing table, **debug ip igrp transaction** can flood the console and make the router unusable. In this case, use **debug ip igrp events** instead to display summary routing information.

Sample Display

Figure 2-95 shows sample **debug ip igrp transaction** output.

Figure 2-95 Sample Debug IP IGRP Transaction Output

```
Router# debug ip igrp transactions
Updates sent to these two source addresses — IGRP: received update from 160.89.80.240 on Ethernet
subnet 160.89.66.0, metric 1300 (neighbor 1200)
subnet 160.89.56.0, metric 8676 (neighbor 8576)
subnet 160.89.48.0, metric 1200 (neighbor 1100)
subnet 160.89.50.0, metric 1300 (neighbor 1200)
subnet 160.89.40.0, metric 8676 (neighbor 8576)
network 192.82.152.0, metric 158550 (neighbor 158450)
network 192.68.151.0, metric 1115511 (neighbor 1115411)
network 150.136.0.0, metric 16777215 (inaccessible)
exterior network 129.140.0.0, metric 9676 (neighbor 9576)
exterior network 140.222.0.0, metric 9676 (neighbor 9576)
IGRP: received update from 160.89.80.28 on Ethernet
subnet 160.89.95.0, metric 180671 (neighbor 180571)
subnet 160.89.81.0, metric 1200 (neighbor 1100)
subnet 160.89.15.0, metric 16777215 (inaccessible)
Updates received from these two destination addresses — IGRP: sending update to 255.255.255.255 via Ethernet0 (160.89.64.31)
subnet 160.89.94.0, metric=847
IGRP: sending update to 255.255.255.255 via Serial1 (160.89.94.31)
subnet 160.89.80.0, metric=16777215
subnet 160.89.64.0, metric=1100
```

Figure 2-95 shows that the router being debugged has received updates from two other routers on the network. The router at source address 160.89.80.240 sent information about ten destinations in the update; the router at source address 160.89.80.28 sent information about three destinations in its update. The router being debugged also sent updates—in both cases to the broadcast address 255.255.255.255 as the destination address.

The first line in Figure 2-95 is self-explanatory.

On the second line in Figure 2-95, the first field refers to the type of destination information: “subnet” (interior), “network” (system), or “exterior” (exterior). The second field is the Internet address of the destination network. The third field is the metric stored in the routing table and the metric advertised by the neighbor sending the information. “Metric... inaccessible” usually means that the neighbor router has put the destination in holddown.

The entries in Figure 2-95 show that the router is sending updates that are similar, except that the numbers in parentheses are the source addresses used in the IP header. A metric of 16777215 is inaccessible.

Other examples of output that the **debug ip igrp transaction** command can produce follow.

The following entry indicates that the routing table was updated and shows the new edition number (97 in this case) to be used in the next IGRP update:

```
IGRP: edition is now 97
```

Entries such as the following occur on startup or when some event occurs such as an interface transitioning or a user manually clearing the routing table:

```
IGRP: broadcasting request on Ethernet0  
IGRP: broadcasting request on Ethernet1
```

The following type of entry can result when routing updates become corrupted between sending and receiving routers:

```
IGRP: bad checksum from 172.21.64.43
```

An entry such as the following should never appear. If it does, the receiving router has a bug in the software or a problem with the hardware. In either case, contact your technical support representative.

```
IGRP: system 45 from 172.21.64.234, should be system 109
```

debug ip mcache

Use the **debug ip mcache** command to display IP multicast fast-switching events. The **no** form of this command disables debugging output.

[no] debug ip mcache [*name* | *address*]

Syntax Description

type (Optional) Interface type.

number (Optional) Interface number.

This command has no arguments or keywords.

Usage Guidelines

Use this command when multicast fast-switching appears not to be functioning.

Sample Display

Figure 2-96 shows sample **debug ip mcache** output when an IP multicast route is cleared.

Figure 2-96 Sample Debug IP Mcache Output

```
router# debug ip mcache

IP multicast fast-switching debugging is on

router#clear ip mroute *

MRC: Build MAC header for (172.31.60.185/32, 224.2.231.173), Ethernet0
MRC: Fast-switch flag for (172.31.60.185/32, 224.2.231.173), off -> on, caller
ip_mroute_replicate-1
MRC: Build MAC header for (172.31.191.10/32, 224.2.127.255), Ethernet0
MRC: Build MAC header for (172.31.60.152/32, 224.2.231.173), Ethernet0
```

Table 2-49 provides explanations for representative lines of the **debug ip mcache** output shown in Figure 2-96.

Table 2-49 Debug IP Mcache Descriptions

Field	Description
MRC	Multicast route cache.
Fast-switch flag	Route is fast-switched.
(<i>address</i> /32)	Host route with 32 bits of mask.
off -> on	State has changed.
caller <i>string</i>	The code function that activated the state change.

Related Commands

debug ip dvmrp

debug ip igmp

debug ip igrp transaction

debug ip mrouting

debug ip sd |

debug ip mpacket

Use the **debug ip mpacket** EXEC command to display IP multicast packets received and transmitted. The **no** form of this command disables debugging output.

[no] debug ip mpacket [detail] [access-list] [group]

Syntax Description

detail	(Optional) Causes the debug ip mpacket command to display IP header information as well as MAC address information.
<i>access-list</i>	(Optional) Access list number.
<i>group</i>	(Optional) Group name or address.

Usage Guidelines

This command displays information for multicast IP packets that are forwarded from this router. By using the *access-list* or *group* argument, you can limit the display to multicast packets from sources described by the access list or a specific multicast group.

Use this command with **debug ip packet** to observe additional packet information.

Note The **debug ip mpacket** command generates lots of messages. Use this command with care so that performance on the network is not affected by the debug message traffic.

Sample Display

Figure 2-97 shows sample **debug ip mpacket** output.

Figure 2-97 Sample Debug IP Mpacket Output

```
router# debug ip mpacket 224.2.0.1

IP: s=10.188.34.54 (Ethernet1), d=224.2.0.1 (Tunnel0), len 88, mforward
IP: s=10.188.34.54 (Ethernet1), d=224.2.0.1 (Tunnel0), len 88, mforward
IP: s=10.188.34.54 (Ethernet1), d=224.2.0.1 (Tunnel0), len 88, mforward
IP: s=10.162.3.27 (Ethernet1), d=224.2.0.1 (Tunnel0), len 68, mforward
```

Table 2-50 defines fields shown in Figure 2-97.

Table 2-50 Debug IP Mpacket Field Descriptions

Field	Description
IP	An IP packet.
<i>s=address</i>	The source address of the packet.
(Ethernet1)	The name of the interface that received the packet.
<i>d=address</i>	The multicast group address that is the destination for this packet.

Table 2-50 Debug IP Mpacket Field Descriptions (Continued)

Field	Description
(Tunnel0)	The outgoing interface for the packet.
len 88	The number of bytes in the packet. This value will vary depending on the application and the media.
mforward	The packet has been forwarded.
not RPF interface	The interface is not a reverse packet forwarding interface. (See debug ip mrouting .)
RPF lookup failed	The reverse packet forwarding lookup failed. (See debug ip mrouting .)

Related Commands

debug ip dvmrp
debug ip igmp
debug ip mrouting
debug ip packet
debug ip sd

debug ip mrouting

Use the **debug ip mrouting** EXEC command to display changes to the IP multicast routing table. The **no** form of this command disables debugging output.

[no] debug ip mrouting [*group*]

Syntax Description

group (Optional) Group name or address to monitor a single group's packet activity.

Usage Guidelines

This command tells when the router has made changes to the mroute table. Use the **debug ip pim** and **debug ip mrouting** commands at the same time to obtain additional multicast routing information. In addition, use the **debug ip igmp** command to see why an mroute message is being displayed.

This command generates a large amount of output. Use the optional *group* to limit the output to a single multicast group.

Sample Display

Figure 2-98 shows sample **debug ip mrouting** output.

Figure 2-98 Sample Debug IP Mrouting Output

```
router# debug ip mrouting 224.2.0.1
IP multicast routing debugging is on

MRT: Delete (10.0.0.0/8, 224.2.0.1)
MRT: Delete (10.4.0.0/16, 224.2.0.1)
MRT: Delete (10.6.0.0/16, 224.2.0.1)
MRT: Delete (10.9.0.0/16, 224.2.0.1)
MRT: Delete (10.16.0.0/16, 224.2.0.1)
MRT: Create (*, 224.2.0.1), if_input NULL
MRT: Create (172.24.15.0/24, 225.2.2.4), if_input Ethernet0, RPF nbr 172.16.61.15
MRT: Create (172.24.39.0/24, 225.2.2.4), if_input Ethernet1, RPF nbr 0.0.0.0
MRT: Create (10.0.0.0/8, 224.2.0.1), if_input Ethernet1, RPF nbr 0.0.0.0
MRT: Create (10.4.0.0/16, 224.2.0.1), if_input Ethernet1, RPF nbr 0.0.0.0
MRT: Create (10.6.0.0/16, 224.2.0.1), if_input Ethernet1, RPF nbr 0.0.0.0
MRT: Create (10.9.0.0/16, 224.2.0.1), if_input Ethernet1, RPF nbr 0.0.0.0
MRT: Create (10.16.0.0/16, 224.2.0.1), if_input Ethernet1, RPF nbr 0.0.0.0
```

Explanations for individual lines of output from Figure 2-98 follow.

The following lines show that multicast IP routes were deleted from the routing table:

```
MRT: Delete (10.0.0.0/8, 224.2.0.1)
MRT: Delete (10.4.0.0/16, 224.2.0.1)
MRT: Delete (10.6.0.0/16, 224.2.0.1)
```

The *,G entry in the following line is always null since it is a *,G. The *,G entries are generally created by receipt of an IGMP host-report from a group member on the directly connected LAN or by a PIM join message (in sparse mode) which this router receives from a router that is sending joins toward the RP. This router will in turn, send a join toward the RP which creates the shared tree (or RP tree).

```
MRT: Create (*, 224.2.0.1), if_input NULL
```

The following lines are an example of creating an S,G entry that show a packet was received on E0. The second line shows a route being created for a source that is on a directly connected LAN. The RPF means “reverse path forwarding,” whereby the router looks up the source address of the multicast packet in the unicast routing table and asks which interface will be used to send a packet to that source.

```
MRT: Create (172.24.15.0/24, 225.2.2.4), if_input Ethernet0, RPF nbr 172.16.61.15
MRT: Create (172.24.39.0/24, 225.2.2.4), if_input Ethernet1, RPF nbr 0.0.0.0
```

The following lines show that multicast IP routes were added to the routing table. Note the 0.0.0.0 as the RPF, which means the route was created by a source that is directly connected to this router.

```
MRT: Create (10.9.0.0/16, 224.2.0.1), if_input Ethernet1, RPF nbr 0.0.0.0
MRT: Create (10.16.0.0/16, 224.2.0.1), if_input Ethernet1, RPF nbr 0.0.0.0
```

If the source is not directly connected, the nbr address shown in these lines will be the address of the router that forwarded the packet to this router.

The shortest path tree state maintained in routers consists of source (S), multicast address (G), outgoing interface (OIF), and incoming interface (IIF). The forwarding information is referred to as the multicast forwarding entry for (S,G).

An entry for a shared tree can match packets from any source for its associated group if the packets come through the proper incoming interface as determined by the RPF lookup. Such an entry is denoted as (*,G). A (*,G) entry keeps the same information a (S,G) entry keeps, except that it saves the rendezvous point (RP) address in place of the source address in sparse mode or 0.0.0.0 in dense mode.

Related Commands

```
debug ip dvmrp
debug ip igmp
debug ip pim
debug ip packet
debug ip sd
```

debug ip nat

Use the **debug ip nat** EXEC command to display information about IP packets translated by the IP network address translation (NAT) feature. The **no** form of this command disables debugging output.

[no] debug ip nat [*access-list* | **detailed**]

Syntax Description

<i>access-list</i>	(Optional) Standard IP access list number. If the datagram is not permitted by the specified access list, the related debugging output is suppressed.
detailed	(Optional) Displays debug information in a detailed format.

Usage Guidelines

The NAT feature reduces the need for unique, registered IP addresses. It can also save private network administrators from having to renumber hosts and routers that do not conform to global IP addressing.

Use the **debug ip nat** command to verify the operation of the NAT feature by displaying information about every packet that is translated by the router. The **debug ip nat detailed** command generates a description of each packet considered for translation. This command also outputs information about certain errors or exceptional conditions, such as the failure to allocate a global address.



Caution Because the **debug ip nat** command generates a significant amount of output, use it only when traffic on the IP network is low, so other activity on the system is not adversely affected.

Sample Display

Figure 2-99 shows sample **debug ip nat** output. In this example, the first two lines show the debugging output that a Domain Name System (DNS) request and reply produced. The remaining lines show the debugging output from a Telnet connection from a host on the inside of the network to a host on the outside of the network. All Telnet packets, except for the first packet, were translated in the fast path, as indicated by the asterisk (*).

Figure 2-99 Sample Debug IP NAT Output

```
router# debug ip nat

NAT: s=192.168.1.95->172.31.233.209, d=172.31.2.132 [6825]
NAT: s=172.31.2.132, d=172.31.233.209->192.168.1.95 [21852]
NAT: s=192.168.1.95->172.31.233.209, d=172.31.1.161 [6826]
NAT*: s=172.31.1.161, d=172.31.233.209->192.168.1.95 [23311]
NAT*: s=192.168.1.95->172.31.233.209, d=172.31.1.161 [6827]
NAT*: s=192.168.1.95->172.31.233.209, d=172.31.1.161 [6828]
NAT*: s=172.31.1.161, d=172.31.233.209->192.168.1.95 [23313]
NAT*: s=172.31.1.161, d=172.31.233.209->192.168.1.95 [23325]
```

Table 2-51 describes the fields and messages shown in Figure 2-99.

Table 2-51 Debug IP NAT Field Descriptions

Field	Description
NAT:	Indicates that the packet is being translated by the network address translation feature. An asterisk (*) indicates the translation is occurring in the fast path. The first packet in a conversation always goes through the slow path (that is, process-switched). The remaining packets go through the fast path if a cache entry exists.
s=192.168.1.95->172.31.233.209	Source address of the packet and how it is being translated.
d=172.31.2.132	Destination address of the packet.
[6825]	IP identification number of the packet. Might be useful in the debugging process to correlate with other packet traces from protocol analyzers.

Figure 2-100 shows sample **debug ip nat detailed** output. In this example, the first two lines show the debugging output that a Domain Name System (DNS) request and reply produced. The remaining lines show the debugging output from a Telnet connection from a host on the inside of the network to a host on the outside of the network. In this example, the inside host 192.168.1.95 was assigned the global address 172.31.233.193.

Figure 2-100 Sample Debug IP NAT Detailed Output

```
router# debug ip nat detailed
NAT: i: udp (192.168.1.95, 1493) -> (172.31.2.132, 53) [22399]
NAT: o: udp (172.31.2.132, 53) -> (172.31.233.193, 1493) [63671]
NAT*: i: tcp (192.168.1.95, 1135) -> (172.31.2.75, 23) [22400]
NAT*: o: tcp (172.31.2.75, 23) -> (172.31.233.193, 1135) [22002]
NAT*: i: tcp (192.168.1.95, 1135) -> (172.31.2.75, 23) [22401]
NAT*: i: tcp (192.168.1.95, 1135) -> (172.31.2.75, 23) [22402]
NAT*: o: tcp (172.31.2.75, 23) -> (172.31.233.193, 1135) [22060]
NAT*: o: tcp (172.31.2.75, 23) -> (172.31.233.193, 1135) [22071]
```

Table 2-52 describes the fields and messages shown in Figure 2-100.

Table 2-52 Debug IP NAT Detailed Field Descriptions

Field	Description
NAT:	Indicates that the packet is being translated by the network address translation feature. An asterisk (*) indicates the translation is occurring in the fast path.
i:	Indicates that the packet is moving from a host inside the network to one outside the network.
o:	Indicates that the packet is moving from a host outside the network to one inside the network.
udp	Protocol of the packet.
(192.168.1.95, 1493) -> (172.31.2.132, 53)	Indicates that the packet is sent from IP address 192.168.1.95 port number 1493 to IP address 172.31.2.132 port number 53.
[22399]	IP identification number of the packet.

debug ip ospf events

Use the **debug ip ospf events** EXEC command to display information on Open Shortest Path First (OSPF)-related events, such as adjacencies, flooding information, designated router selection, and shortest path first (SPF) calculation. The **no** form of this command disables debugging output.

[no] debug ip ospf events

Sample Display

Figure 2-101 shows sample **debug ip ospf events** output.

Figure 2-101 Sample Debug IP OSPF Events Output

```
router# debug ip ospf events

OSPF:hello with invalid timers on interface Ethernet0
hello interval received 10 configured 10
net mask received 255.255.255.0 configured 255.255.255.0
dead interval received 40 configured 30
```

The **debug ip ospf events** output shown in Figure 2-101 might appear if any of the following occurs:

- The IP subnet masks for routers on the same network do not match.
- The OSPF hello interval for the router does not match that configured for a neighbor.
- The OSPF dead interval for the router does not match that configured for a neighbor.

If a router configured for OSPF routing is not seeing an OSPF neighbor on an attached network, do the following:

- Make sure that both routers have been configured with the same IP mask, OSPF hello interval, and OSPF dead interval.
- Make sure that both neighbors are part of the same area type.

In the following example line, the neighbor and this router are not part of a stub area (that is, one is a part of a transit area and the other is a part of a stub area, as explained in RFC 1247).

```
OSPF: hello packet with mismatched E bit
```

Related Command

debug ip ospf packet

debug ip ospf packet

Use the **debug ip ospf packet** EXEC command to display information about each Open Shortest Path First (OSPF) packet received. The **no** form of this command disables debugging output.

[no] debug ip ospf packet

Sample Display

Figure 2-102 shows sample **debug ip ospf packet** output.

Figure 2-102 Sample Debug IP OSPF Packet Output

```
router# debug ip ospf packet

OSPF: rcv. v:2 t:1 l:48 rid:200.0.0.117
      aid:0.0.0.0 chk:6AB2 aut:0 auk:
```

The **debug ip ospf packet** command produces one set of information for each packet received. The output varies slightly depending on which authentication is used. Figure 2-103 shows sample **debug ip ospf packet** output when MD5 authentication is used.

Figure 2-103 Sample Debug IP OSPF Packet Output—MD5 Authentication

```
router# debug ip ospf packet

OSPF: rcv. v:2 t:1 l:48 rid:200.0.0.116
      aid:0.0.0.0 chk:0 aut:2 keyid:1 seq:0x0
```

Table 2-53 describes the fields shown in Figure 2-102 and Figure 2-103.

Table 2-53 Debug IP OSPF Packet Field Descriptions

Field	Description
v:	OSPF version.
t:	OSPF packet type. Possible packet types follow: 1—Hello 2—Data description 3—Link state request 4—Link state update 5—Link state acknowledgment
l:	OSPF packet length in bytes.
rid:	OSPF router ID.
aid:	OSPF area ID.
chk:	OSPF checksum.
aut:	OSPF authentication type. Possible authentication types follow: 0—No authentication 1—Simple password 2—MD5

Table 2-53 Debug IP OSPF Packet Field Descriptions (Continued)

Field	Description
auk:	OSPF authentication key.
keyid:	MD5 key ID.
seq:	Sequence number.

Related Command
debug ip ospf events

debug ip packet

Use the **debug ip packet** EXEC command to display general IP debugging information and IP security option (IPSO) security transactions. The **no** form of this command disables debugging output.

[no] debug ip packet [*access-list-number*]

Syntax Description

access-list-number (Optional) IP access list number that you can specify. If the datagram is not permitted by that access list, the related debugging output is suppressed.

Usage Guidelines

If a communication session is closing when it should not be, an end-to-end connection problem can be the cause. The **debug ip packet** command is useful for analyzing the messages traveling between the local and remote hosts.

IP debugging information includes packets received, generated, and forwarded. Fast-switched packets do not generate messages.

IPSO security transactions include messages that describe the cause of failure each time a datagram fails a security test in the system. This information is also sent to the sending host when the router configuration allows it.

Note Because the **debug ip packet** command generates a significant amount of output, use it only when traffic on the IP network is low, so other activity on the system is not adversely affected.

Sample Display

Figure 2-104 shows sample **debug ip packet** output.

Figure 2-104 Sample Debug IP Packet Output

```
router# debug ip packet
IP: s=172.16.13.44 (Fddi0), d=10.125.254.1 (Serial2), g=172.16.16.2, forward
IP: s=172.16.1.57 (Ethernet4), d=10.36.125.2 (Serial2), g=172.16.16.2, forward
IP: s=172.16.1.6 (Ethernet4), d=255.255.255.255, rcvd 2
IP: s=172.16.1.55 (Ethernet4), d=172.16.2.42 (Fddi0), g=172.16.13.6, forward
IP: s=172.16.89.33 (Ethernet2), d=10.130.2.156 (Serial2), g=172.16.16.2, forward
IP: s=172.16.1.27 (Ethernet4), d=172.16.43.126 (Fddi1), g=172.16.23.5, forward
IP: s=172.16.1.27 (Ethernet4), d=172.16.43.126 (Fddi0), g=172.16.13.6, forward
IP: s=172.16.20.32 (Ethernet2), d=255.255.255.255, rcvd 2
IP: s=172.16.1.57 (Ethernet4), d=10.36.125.2 (Serial2), g=172.16.16.2, access denied
```

Figure 2-104 shows two types of messages that the **debug ip packet** command can produce; the first line of output describes an IP packet that the router forwards, and the third line of output describes a packet that is destined for the router. In the third line of output, “rcvd 2” indicates that the router decided to receive the packet.

Table 2-54 describes the fields shown in the first line of Figure 2-104.

Table 2-54 Debug IP Packet Field Descriptions

Field	Description
IP:	Indicates that this is an IP packet.
s = 172.16.13.44 (Fddi0)	Indicates the source address of the packet and the name of the interface that received the packet.
d = 10.125.254.1 (Serial2)	Indicates the destination address of the packet and the name of the interface (in this case, S2) through which the packet is being sent out on the network.
g = 172.16.16.2	Indicates the address of the next hop gateway.
forward	Indicates that the router is forwarding the packet. If a filter denies a packet, "access denied" replaces "forward," as shown in the last line of output in Figure 2-104.

The calculation on whether to send a security error message can be somewhat confusing. It depends upon both the security label in the datagram and the label of the incoming interface. First, the label contained in the datagram is examined for anything obviously wrong. If nothing is wrong, assume it to be correct. If there is something wrong, the datagram is treated as *unclassified genser*. Then the label is compared with the interface range, and the appropriate action is taken as Table 2-55 describes.

Table 2-55 Security Actions

Classification	Authorities	Action Taken
Too low	Too low	No Response
	Good	No Response
	Too high	No Response
In range	Too low	No Response
	Good	Accept
	Too high	Send Error
Too high	Too low	No Response
	In range	Send Error
	Too high	Send Error

The security code can only generate a few types of ICMP error messages. The only possible error messages and their meanings follow:

- "ICMP Parameter problem, code 0"—Error at pointer
- "ICMP Parameter problem, code 1"—Missing option
- "ICMP Parameter problem, code 2"—See Note that follows
- "ICMP Unreachable, code 10"—Administratively prohibited

Note The message “ICMP Parameter problem, code 2” identifies a specific error that occurs in the processing of a datagram. This message indicates that the router received a datagram containing a maximum length IP header but no security option. After being processed and routed to another interface, it is discovered that the outgoing interface is marked with “add a security label.” Since the IP header is already full, the system cannot add a label and must drop the datagram and return an error message.

When an IP packet is rejected due to an IP security failure, an audit message is sent via DNSIX NAT. Also, any **debug ip packet** output is appended to include a description of the reason for rejection. This description can be any of the following:

- No basic
- No basic, no response
- Reserved class
- Reserved class, no response
- Class too low, no response
- Class too high
- Class too high, bad authorities, no response
- Unrecognized class
- Unrecognized class, no response
- Multiple basic
- Multiple basic, no response
- Authority too low, no response
- Authority too high
- Compartment bits not dominated by maximum sensitivity level
- Compartment bits do not dominate minimum sensitivity level
- Security failure: extended security disallowed
- NLESO source appeared twice
- ESO source not found
- Postroute, failed xfc out
- No room to add IPSO

debug ip pim

Use the **debug ip pim EXEC** command to display Protocol Independent Multicast (PIM) packets received and transmitted as well as PIM related events. The **no** form of this command disables debugging output.

[no] debug ip pim [*group*]

Syntax Description

group (Optional) Group name or address to monitor a single group's packet activity.

Usage Guidelines

PIM uses IGMP packets to communicate between routers and advertise reachability information.

Use this command with **debug ip igmp** and **debug ip mrouting** to observe additional multicast routing information.

Sample Display

Figure 2-105 shows sample **debug ip pim** output.

Figure 2-105 Sample Debug IP PIM Output

```
router# debug ip pim 224.2.0.1

PIM: Received Join/Prune on Ethernet1 from 172.24.37.33
PIM: Received Join/Prune on Ethernet1 from 172.24.37.33
PIM: Received Join/Prune on Tunnel0 from 10.3.84.1
PIM: Received Join/Prune on Ethernet1 from 172.24.37.33
PIM: Received Join/Prune on Ethernet1 from 172.24.37.33
PIM: Received RP-Reachable on Ethernet1 from 172.16.20.31
PIM: Update RP expiration timer for 224.2.0.1
PIM: Forward RP-reachability packet for 224.2.0.1 on Tunnel0
PIM: Received Join/Prune on Ethernet1 from 172.24.37.33
PIM: Prune-list (10.221.196.51/32, 224.2.0.1)
PIM: Set join delay timer to 2 seconds for (10.221.0.0/16, 224.2.0.1) on Ethernet1
PIM: Received Join/Prune on Ethernet1 from 172.24.37.6
PIM: Received Join/Prune on Ethernet1 from 172.24.37.33
PIM: Received Join/Prune on Tunnel0 from 10.3.84.1
PIM: Join-list: (*, 224.2.0.1) RP 172.16.20.31
PIM: Add Tunnel0 to (*, 224.2.0.1), Forward state
PIM: Join-list: (10.0.0.0/8, 224.2.0.1)
PIM: Add Tunnel0 to (10.0.0.0/8, 224.2.0.1), Forward state
PIM: Join-list: (10.4.0.0/16, 224.2.0.1)
PIM: Prune-list (172.24.84.16/28, 224.2.0.1) RP-bit set RP 172.24.84.16
PIM: Send Prune on Ethernet1 to 172.24.37.6 for (172.24.84.16/28, 224.2.0.1), RP
PIM: For RP, Prune-list: 10.9.0.0/16
PIM: For RP, Prune-list: 10.16.0.0/16
PIM: For RP, Prune-list: 10.49.0.0/16
PIM: For RP, Prune-list: 10.84.0.0/16
PIM: For RP, Prune-list: 10.146.0.0/16
PIM: For 10.3.84.1, Join-list: 172.24.84.16/28
PIM: Send periodic Join/Prune to RP via 172.24.37.6 (Ethernet1)
```

Explanations for individual lines of output from Figure 2-105 follow.

The following lines appear periodically when PIM is running in sparse mode and indicate to this router which multicast groups and multicast sources other routers are interested in:

```
PIM: Received Join/Prune on Ethernet1 from 172.24.37.33
PIM: Received Join/Prune on Ethernet1 from 172.24.37.33
```

The following lines appear when a rendezvous point (RP) message is received and the RP timer is reset. The expiration timer sets a checkpoint to make sure the RP still exists; otherwise a new RP must be discovered:

```
PIM: Received RP-Reachable on Ethernet1 from 172.16.20.31
PIM: Update RP expiration timer for 224.2.0.1
PIM: Forward RP-reachability packet for 224.2.0.1 on Tunnel0
```

The prune-list message in the following line states that this router is not interested in the source address information. The prune message tells an upstream router to stop forwarding multicast packets from this source.

```
PIM: Prune-list (10.221.196.51/32, 224.2.0.1)
```

In the following line, a second router on the network wants to override the prune message that the upstream router just received. The timer is set at a random value so that if there are additional routers on the network that still want to receive multicast packets for the group, only one will actually send the message. The other routers will receive the join message and then suppress sending their own message.

```
PIM: Set join delay timer to 2 seconds for (10.221.0.0/16, 224.2.0.1) on Ethernet1
```

In the following line, a join message is sent towards the RP for all sources:

```
PIM: Join-list: (*, 224.2.0.1) RP 172.16.20.31
```

In the following lines, the interface is being added to the outgoing interface (OIF) of the *,G and S,G mroute table entry so that packets from the source will be forwarded out that particular interface:

```
PIM: Add Tunnel0 to (*, 224.2.0.1), Forward state
PIM: Add Tunnel0 to (10.0.0.0/8, 224.2.0.1), Forward state
```

The following line appears in sparse mode only. There are two trees on which data may be received: the RP tree and the source tree. In dense mode there is no RP. After the source and the receiver have discovered one another at the RP, the first-hop router for the receiver will usually join to the source tree rather than the RP tree:

```
PIM: Prune-list (172.24.84.16/28, 224.2.0.1) RP-bit set RP 172.24.84.16
```

The Send Prune message in the next line shows that a router is sending a message to a second router saying that the first router no longer wants to receive multicast packets for the S,G. The “RP” at the end of the message indicates that the router is pruning the RP tree and is most likely joining the source tree, although the router may not have downstream members for the group or downstream routers with members of the group. The output shows which specific sources this router no longer wants to receive multicast from.

```
PIM: Send Prune on Ethernet1 to 172.24.37.6 for (172.24.84.16/28, 224.2.0.1), RP
```

The following lines indicate a prune message is sent toward the RP so that router can join the source tree rather than the RP tree:

```
PIM: For RP, Prune-list: 10.9.0.0/16
PIM: For RP, Prune-list: 10.16.0.0/16
PIM: For RP, Prune-list: 10.49.0.0/16
```

In the following line, a periodic message is sent towards the RP. The default period is once per minute. Prune and join messages are sent toward the RP or source rather than directly to the RP or source. It is the responsibility of the next-hop router to take proper action with this message, such as continuing to forward it to the next router in the tree.

```
PIM: Send periodic Join/Prune to RP via 172.24.37.6 (Ethernet1)
```

Related Commands

debug ip dvmrp

debug ip igmp

debug ip igmp transaction

debug ip mrouting

debug ip sd

debug ip pim auto-rp

Use the **debug ip pim auto-rp EXEC** command to display the contents of each Protocol Independent Multicast (PIM) packet used in the automatic discovery of group-to-rendezvous point (RP) mapping as well as the actions taken on the address-to-RP mapping database. The **no** form of this command disables debugging output.

[no] debug ip pim auto-rp

Sample Display

Figure 2-106 shows sample **debug ip pim auto-rp** output.

Figure 2-106 Sample Debug IP PIM Auto-RP Output

```
router# debug ip pim auto-rp

Auto-RP: Received RP-announce, from 172.31.214.66, RP_cnt 1, holdtime 180 secs
Auto-RP: update (192.168.248.0/24, RP:172.31.214.66)
Auto-RP: Build RP-Discovery packet
Auto-RP: Build mapping (192.168.248.0/24, RP:172.31.214.66),
Auto-RP: Build mapping (192.168.250.0/24, RP:172.31.214.26).
Auto-RP: Build mapping (192.168.254.0/24, RP:172.31.214.2).
Auto-RP: Send RP-discovery packet (3 RP entries)
Auto-RP: Build RP-Announce packet for 172.31.214.2
Auto-RP: Build announce entry for (192.168.254.0/24)
Auto-RP: Send RP-Announce packet, IP source 172.31.214.2, ttl 8
```

Explanations for individual lines of output from Figure 2-106 follow.

The first two lines show a packet received from 172.31.214.66 announcing that it is the rendezvous point (RP) for the groups in 192.168.248.0/24. This announcement contains one RP address and is valid for 180 seconds. The RP-mapping agent then updates its mapping database to include the new information.

```
Auto-RP: Received RP-announce, from 172.31.214.66, RP_cnt 1, holdtime 180 secs
Auto-RP: update (192.168.248.0/24, RP:172.31.214.66)
```

In the next five lines, the router creates an RP-discovery packet containing three RP mapping entries. The packet is sent to the well-known CISCO-RP-DISCOVERY group address (224.0.1.40).

```
Auto-RP: Build RP-Discovery packet
Auto-RP: Build mapping (192.168.248.0/24, RP:172.31.214.66),
Auto-RP: Build mapping (192.168.250.0/24, RP:172.31.214.26).
Auto-RP: Build mapping (192.168.254.0/24, RP:172.31.214.2).
Auto-RP: Send RP-discovery packet (3 RP entries)
```

The final three lines show the router announcing that it intends to be an RP for the groups in 192.168.254.0/24. Only routers inside the scope ttl 8 receive the advertisement and use the RP for these groups.

```
Auto-RP: Build RP-Announce packet for 172.31.214.2
Auto-RP: Build announce entry for (192.168.254.0/24)
Auto-RP: Send RP-Announce packet, IP source 172.31.214.2, ttl 8
```

Figure 2-107 shows sample **debug ip pim auto-rp** output when a router receives an update. In this example, the packet contains three group-to-RP mappings, which are valid for 180 seconds. The RP-mapping agent then updates its mapping database to include the new information.

Figure 2-107 Sample Debug IP PIM Auto-RP Output—Router Receiving Update

```
router# debug ip pim auto-rp

Auto-RP: Received RP-discovery, from 172.31.214.17, RP_cnt 3, holdtime 180 secs
Auto-RP: update (192.168.248.0/24, RP:172.31.214.66)
Auto-RP: update (192.168.250.0/24, RP:172.31.214.26)
Auto-RP: update (192.168.254.0/24, RP:172.31.214.2)
```

debug ip policy

Use the **debug ip policy** EXEC command to display IP policy routing packet activity. The **no** form of this command disables debugging output.

[no] debug ip policy

Usage Guidelines

After you configure IP policy routing with the **ip policy** and **route map** commands, use the **debug ip policy** command to ensure that the IP policy is configured correctly.

Policy routing looks at various parts of the packet and then routes the packet based on certain user-defined attributes in the packet.

The **debug ip policy** command helps you determine what policy routing is doing. It displays information about whether a packet matches the criteria, and if so, the resulting routing information for the packet.



Caution Because the **debug ip policy** command generates a significant amount of output, use it only when traffic on the IP network is low, so other activity on the system is not adversely affected.

You can also use the **show ip local policy** command to obtain additional information.

Sample Display

Figure 2-108 shows sample **debug ip policy** output. Line 1 indicates that a packet with the given source and destination addresses matched a policy. Line 2 indicates the clause in the route map that the packet matched. In this case, the packet matches clause 20 in the route map. Line 3 indicates that a second packet did not match the policy.

Figure 2-108 Sample Debug IP Policy Output

```
router# debug ip policy
IP: s=172.16.232.150 (local), d=172.16.2.75, len 100, policy match
IP: route map equal, item 20, permit
IP: s=172.16.232.150 (local), d=172.16.2.75, len 200, policy rejected -- normal
forwarding
```

debug ip rip

Use the **debug ip rip** EXEC command to display information on RIP routing transactions. The **no** form of this command disables debugging output.

[no] debug ip rip

Sample Display

Figure 2-109 shows sample **debug ip rip** output.

Figure 2-109 Sample Debug IP RIP Output

```

router# debug ip rip
pdates
received — RIP: received update from 10.89.80.28 on Ethernet0
om this   10.89.95.0 in 1 hops
source    10.89.81.0 in 1 hops
address   10.89.66.0 in 2 hops
          172.31.0.0 in 16 hops (inaccessible)
          0.0.0.0 in 7 hop
pdates
sent to   — RIP: sending update to 255.255.255.255 via Ethernet0 (10.89.64.31)
these two subnet 10.89.94.0, metric 1
estimation 172.31.0.0 in 16 hops (inaccessible)
addresses — RIP: sending update to 255.255.255.255 via Serial1 (10.89.94.31)
          subnet 10.89.64.0, metric 1
          subnet 10.89.66.0, metric 3
          172.31.0.0 in 16 hops (inaccessible)
          default 0.0.0.0, metric 8

```

95250

Figure 2-109 shows that the router being debugged has received updates from one router at source address 160.89.80.28. That router sent information about five destinations in the routing table update. Notice that the fourth destination address in the update—131.108.0.0—is inaccessible because it is more than 15 hops away from the router sending the update. The router being debugged also sent updates, in both cases to broadcast address 255.255.255.255 as the destination.

The first line in Figure 2-109 is self-explanatory.

The second line in Figure 2-109 is an example of a routing table update. It shows how many hops a given Internet address is from the router.

The entries in Figure 2-109 show that the router is sending updates that are similar, except that the number in parentheses is the source address encapsulated into the IP header.

Examples of additional output that the **debug ip rip** command can generate follow.

Entries such as the following appear at startup or when an event occurs such as an interface transitioning or a user manually clearing the routing table:

```

RIP: broadcasting general request on Ethernet0
RIP: broadcasting general request on Ethernet1

```

The following line is self-explanatory:

```

RIP: received request from 160.89.80.207 on Ethernet0

```

An entry such as the following is most likely caused by a malformed packet from the transmitter:

```

RIP: bad version 128 from 160.89.80.43

```

debug ip routing

Use the **debug ip routing EXEC** command to display information on Routing Information Protocol (RIP) routing table updates and route-cache updates. The **no** form of this command disables debugging output.

[no] debug ip routing

Sample Display

Figure 2-110 shows sample **debug ip routing** output.

Figure 2-110 Sample Debug IP Routing Output

```
router# debug ip routing

RT: add 172.25.168.0 255.255.255.0 via 172.24.76.30, igrp metric [100/3020]
RT: metric change to 172.25.168.0 via 172.24.76.30, igrp metric [100/3020]
    new metric [100/2930]
IP: cache invalidation from 0x115248 0x1378A, new version 5736
RT: add 172.26.219.0 255.255.255.0 via 172.24.76.30, igrp metric [100/16200]
RT: metric change to 172.26.219.0 via 172.24.76.30, igrp metric [100/16200]
    new metric [100/10816]
RT: delete route to 172.26.219.0 via 172.24.76.30, igrp metric [100/10816]
RT: no routes to 172.26.219.0, entering holddown
IP: cache invalidation from 0x115248 0x1378A, new version 5737
RT: 172.26.219.0 came out of holddown
RT: garbage collecting entry for 172.26.219.0
IP: cache invalidation from 0x115248 0x1378A, new version 5738
RT: add 172.26.219.0 255.255.255.0 via 172.24.76.30, igrp metric [100/10816]
RT: delete route to 172.26.219.0 via 172.24.76.30, igrp metric [100/10816]
RT: no routes to 172.26.219.0, entering holddown
IP: cache invalidation from 0x115248 0x1378A, new version 5739
RT: 172.26.219.0 came out of holddown
RT: garbage collecting entry for 172.26.219.0
IP: cache invalidation from 0x115248 0x1378A, new version 5740
RT: add 172.26.219.0 255.255.255.0 via 172.24.76.30, igrp metric [100/16200]
RT: metric change to 172.26.219.0 via 172.24.76.30, igrp metric [100/16200]
    new metric [100/10816]
RT: delete route to 172.26.219.0 via 172.24.76.30, igrp metric [100/10816]
RT: no routes to 172.26.219.0, entering holddown
IP: cache invalidation from 0x115248 0x1378A, new version 5741
```

Explanations for representative lines of output in Figure 2-110 follow.

In the following lines, a newly created entry has been added to the IP routing table. The “metric change” indicates that this entry existed previously, but its metric changed and the change was reported by means of IGRP. The metric could also be reported via RIP, OSPF, or another IP routing protocol. The numbers inside the brackets report the administrative distance and the actual metric.

“Cache invalidation” means that the fast switching cache was invalidated due to a routing table change. “New version” is the version number of the routing table. When the routing table changes, this number is incremented. The hexadecimal numbers are internal numbers that vary from version to version and software load to software load.

```
RT: add 172.25.168.0 255.255.255.0 via 172.24.76.30, igrp metric [100/3020]
RT: metric change to 172.25.168.0 via 172.24.76.30, igrp metric [100/3020]
    new metric [100/2930]
IP: cache invalidation from 0x115248 0x1378A, new version 5736
```

In the following output, the “holddown” and “cache invalidation” lines are displayed. Most of the distance vector routing protocols use “holddown” to avoid typical problems like counting to infinity and routing loops. If you look at the output of **show ip protocols** you will see what the timer values are for “holddown” and “cache invalidation”. “Cache invalidation” corresponds to “came out of holddown”. “Delete route” is triggered when a better path comes along. It gets rid of the old inferior path.

```
RT: delete route to 172.26.219.0 via 172.24.76.30, igrp metric [100/10816]
RT: no routes to 172.26.219.0, entering holddown
IP: cache invalidation from 0x115248 0x1378A, new version 5737
RT: 172.26.219.0 came out of holddown
```

debug ip rsvp

Use the **debug ip rsvp** EXEC command to enable logging of significant Resource Reservation Protocol (RSVP) events. The **no** form of this command disables debugging output.

[no] debug ip rsvp [detail [access-list]]

Syntax Description

detail	(Optional) Displays debug information in a detailed format.
<i>access-list</i>	(Optional) Standard IP access list number. If the datagram is not permitted by the specified access list, the related debugging output is suppressed.

Usage Guidelines

The RSVP protocol permits end systems to request quality of service (QoS) guarantees from the network.

The **debug ip rsvp** command displays recognition of new senders, subsequent discontinuation of senders, and installation and removal of reservations.

The **detail** option displays recognition of new senders, subsequent discontinuation of senders, installation and removal of reservations, and messages sent and received that are permitted by the specified access list, or all messages if no access list is specified. The output from **detail** option is the same as the **debug ip rsvp** command but it also includes a hexadecimal-dump of the contents of the messages.

Sample Displays

In the following message, the router received a path message for destination 192.168.253.10 on interface Ethernet 0/2. The message was sent by 172.31.215.9—this indicates the previous hop and not the source.

```
RSVP: PATH message for 192.168.253.10(Ethernet0/2) from 172.31.215.9
```

In the following two messages, the router is sending a path message to destination 192.168.253.10 on interface Ethernet 0/0. The second message is a reassurance that the packet is sent.

```
RSVP: send path multicast about 192.168.253.10 on Ethernet0/0
RSVP: IP to 192.168.253.10 length=52 checksum=720C Ethernet0/0
```

In the following message, the router received an reservation (RESV) message for destination 192.168.253.10 on interface serial 0. The message was sent by 172.31.215.113, which is the next hop toward the destination.

```
RSVP: RESV message for 192.168.253.10(Serial0) from 172.31.215.113
```

In the following messages, the router is removing the sender state for destination 1.1.1.1. The destination port is (1234), the source is 172.31.215.11, and the protocol and source port are (17:1234). Removing the sender state triggers a teardown message to be sent toward destination 1.1.1.1 on interface Ethernet 0/2 (located in the routing table).

```
RSVP: remove Ethernet0/1 PATH 1.1.1.1(1234) <- 172.31.215.11(17:1234)
RSVP: send path teardown multicast about 1.1.1.1 on Ethernet0/2
```

The following messages occur when a reservation (RESV) message is received but no sender state (path message) exists that matches it. In this case, the router drops the reservation and sends a reservation error message to destination 192.168.253.10.

```
RSVP: RESV message for 192.168.253.10(Serial0) from 172.31.215.113
RSVP: send reservation error to 172.31.215.113 about 192.168.253.10
RSVP: IP to 172.31.215.113 length=40 checksum=1A84 if Serial0
RSVP RESV: no path information for 192.168.253.10
```

In the following messages, the router tears down a reservation because it received a teardown message or because the message timed out. Tearing down a reservation triggers the removal of the existing installed reservation, as shown in the second message. The teardown also triggers the removal of this reservation from upstream routers. After the router removes the reservation, an RESV teardown message is sent to the source to tear down the reservation on the next hop, as shown in the fourth message. This process continues until the source is reached.

```
RSVP: RESV TEAR message for 192.168.253.10(Ethernet0/0) from 172.31.215.98
RSVP: remove Ethernet0/0 RESV 192.168.253.10(18004) <- 172.31.60.189(17:18004)
RSVP: remove Ethernet0/2 RESV request 192.168.253.10(18004) <- 172.31.60.189(17:18004)
RSVP: send reservation teardown to 172.31.215.9 about 192.168.253.10
```

In the following messages, the router received and accepted a new reservation request. This request triggers the router to send a reservation request message to the source.

```
RSVP: Reservation is new
RSVP: start requesting 30 kbps SE reservation for 172.31.60.189(18004) UDP->
192.168.253.10(18004) on Ethernet0 neighbor 172.31.215.97
```

In the following message, the router received an RSVP message on an interface with the RSVP feature disabled:

```
RSVP: Input packet while RSVP disabled on Serial2/0
```

debug ip sd

Use the **debug ip sd** command to display all session directory (SD) announcements received. The **no** form of this command disables debugging output.

[no] debug ip sd

Usage Guidelines

This command shows session directory announcements for multicast IP. Use it to observe multicast activity.

Sample Display

Figure 2-111 shows sample **debug ip sd** output.

Figure 2-111 Sample Debug IP SD Output

```
router# debug ip sd

SD: Announcement from 172.31.58.81 on Serial0.1, 146 bytes
s=*cisco: CBONE Audio
i=cisco internal-only audio conference
o=dino@dino-ss20.cisco.com
c=224.0.255.1 16 2891478496 2892688096
m=audio 31372 1700

SD: Announcement from 172.22.246.68 on Serial0.1, 147 bytes
s=IMS: U.S. Senate
i=U.S. Senate at http://town.hall.org/radio/live.html
o=carl@also.radio.com
c=224.2.252.231 95 0 0
m=audio 36572 2642
a=fmt:gsm
```

Table 2-56 provides explanations for representative lines of the **debug ip sd** output shown in Figure 2-111.

Table 2-56 Debug IP SD Output Descriptions

Field	Description
SD	Session directory event.
Announcement from	Address sending the SD announcement.
on Serial0.1	Interface receiving the announcement.
146 bytes	Size of the announcement event.
s=	Session name being advertised.
i=	Information providing a descriptive name for the session.
o=	Origin of the session, either an IP address or a name.
c=	Connect description showing address and number of hops.
m=	Media description that includes media type, port number, and ID.

Related Commands

- debug ip dvmrp**
- debug ip igmp**
- debug ip mcache**
- debug ip mrouting**
- debug ip pim**

debug ip security

Use the **debug ip security** EXEC command to display IP security option processing. The **no** form of this command disables debugging output.

[no] debug ip security

Usage Guidelines

The **debug ip security** command displays information for both basic and extended IP security options. For interfaces where **ip security** is configured, each IP packet processed for that interface results in debugging output regardless of whether the packet contains IP security options. IP packets processed for other interfaces that also contain IP security information also trigger debugging output. Some additional IP security debugging information is also controlled by the **debug ip packet** EXEC command.

Note Because the **debug ip security** command generates a significant amount of output for every IP packet processed, use it only when traffic on the IP network is low, so other activity on the system is not adversely affected.

Sample Display

Figure 2-112 shows sample **debug ip security** output.

Figure 2-112 Sample Debug IP Security Output

```
router# debug ip security

IP Security: src 172.24.72.52 dst 172.24.72.53, number of BSO 1
  idb: NULL
  pak: insert (0xFF) 0x0
IP Security: BSO postroute: SECINSERT changed to secret (0x5A) 0x10
IP Security: src 172.24.72.53 dst 172.24.72.52, number of BSO 1
  idb: secret (0x6) 0x10 to secret (0x6) 0x10, no implicit
  def secret (0x6) 0x10
  pak: secret (0x5A) 0x10
IP Security: checking BSO 0x10 against [0x10 0x10]
IP Security: classified BSO as secret (0x5A) 0x10
```

Table 2-57 describes significant fields shown in Figure 2-112.

Table 2-57 Debug IP Security Field Descriptions

Field	Description
number of BSO	Indicates the number of basic security options found in the packet.
idb	Provides information on the security configuration for the incoming interface.
pak	Provides information on the security classification of the incoming packet.
src	Indicates the source IP address.
dst	Indicates the destination IP address.

Explanations for representative lines of output in Figure 2-112 follow.

The following line indicates that the packet was locally generated, and it has been classified with the internally significant security level “insert” (0xff) and authority 0x0:

```
idb: NULL
pak: insert (0xff) 0x0
```

The following line indicates that the packet was received via an interface with dedicated IP security configured. Specifically, the interface is configured at security level “secret” and with authority information of 0x0. The packet itself was classified at level “secret” (0x5a) and authority 0x10.

```
idb: secret (0x6) 0x10 to secret (0x6) 0x10, no implicit
     def secret (0x6) 0x10
pak: secret (0x5A) 0x10
```

debug ip tcp driver

Use the **debug ip tcp driver EXEC** command to display information on Transmission Control Protocol (TCP) driver events; for example, connections opening or closing, or packets being dropped because of full queues. The **no** form of this command disables debugging output.

[no] debug ip tcp driver

Usage Guidelines

The TCP driver is the process that the router software uses to send packet data over a TCP connection. Remote source-route bridging, STUN (serial tunneling), and X.25 switching currently use the TCP driver.

Using the **debug ip tcp driver** command together with the **debug ip tcp driver-pak** command provides the most verbose debugging output concerning TCP driver activity.

Sample Display

Figure 2-113 shows sample **debug ip tcp driver** output.

Figure 2-113 Sample Debug IP TCP Driver Output

```
router# debug ip tcp driver

TCPDRV359CD8: Active open 172.21.80.26:0 --> 172.21.80.25:1996 OK, lport 36628
TCPDRV359CD8: enable tcp timeouts
TCPDRV359CD8: 172.21.80.26:36628 --> 172.21.80.25:1996 Abort
TCPDRV359CD8: 172.21.80.26:36628 --> 172.21.80.25:1996 DoClose tcp abort
```

Explanations for individual lines of output from Figure 2-113 follow.

Table 2-58 describes the fields in the first line of output.

Table 2-58 Debug IP TCP Driver Field Descriptions

Field	Description
TCPDRV359CD8:	Unique identifier for this instance of TCP driver activity.
Active open 172.21.80.26	Indication that the router at IP address 172.21.80.26 has initiated a connection to another router.
:0	The TCP port number the initiator of the connection uses to indicate that any port number can be used to set up a connection.
--> 172.21.80.25	The IP address of the remote router to which the connection has been initiated.
:1996	The TCP port number that the initiator of the connection is requesting that the remote router use for the connection. (1996 is a private TCP port number reserved in this implementation for remote source-route bridging.)
OK,	Indication that the connection has been established. If the connection has not been established, this field and the following field do not appear in this line of output.
lport 36628	The TCP port number that has actually been assigned for the initiator to use for this connection.

The following line indicates that the TCP driver user (remote source-route bridging, in this case) will allow TCP to drop the connection if excessive retransmissions occur:

```
TCPDRV359CD8: enable tcp timeouts
```

The following line indicates that the TCP driver user (in this case, remote source-route bridging) at IP address 172.21.80.26 (and using TCP port number 36628) is requesting that the connection to IP address 172.21.80.25 using TCP port number 1996 be aborted:

```
TCPDRV359CD8: 172.21.80.26:36628 --> 172.21.80.25:1996 Abort
```

The following line indicates that this connection was in fact closed due to an abnormal termination:

```
TCPDRV359CD8: 172.21.80.26:36628 --> 172.21.80.25:1996 DoClose tcp abort
```

debug ip tcp driver-pak

Use the **debug ip tcp driver-pak** EXEC command to display information on every operation that the Transmission Control Protocol (TCP) driver performs. The **no** form of this command disables debugging output.

[no] debug ip tcp driver-pak

Usage Guidelines

This command turns on a verbose debugging by logging at least one debugging message for every packet sent or received on the TCP driver connection.

The TCP driver is the process that the router software uses to send packet data over a TCP connection. Remote source-route bridging, STUN (serial tunneling), and X.25 switching currently use the TCP driver.

To observe the context within which certain **debug ip tcp driver-pak** messages occur, turn on this command in conjunction with the **debug ip tcp driver** command.

Note Because the **debug ip tcp driver-pak** command generates so many messages, use it only on lightly loaded systems. This command not only places a significant load on the system processor, but it may even change the symptoms of any unexpected behavior that occur.

Sample Display

Figure 2-114 shows sample **debug ip tcp driver-pak** output.

Figure 2-114 Sample Debug IP TCP Driver-Pak Output

```
router# debug ip tcp driver-pak

TCPDRV359CD8: send 2E8CD8 (len 26) queued
TCPDRV359CD8: output pak 2E8CD8 (len 26) (26)
TCPDRV359CD8: readf 42 bytes (Thresh 16)
TCPDRV359CD8: readf 26 bytes (Thresh 16)
TCPDRV359CD8: readf 10 bytes (Thresh 10)
TCPDRV359CD8: send 327E40 (len 4502) queued
TCPDRV359CD8: output pak 327E40 (len 4502) (4502)
```

Explanations for individual lines of output from Figure 2-114 follow.

Table 2-59 describes the fields shown in the first line of output.

Table 2-59 Debug TCP Driver-Pak Field Descriptions

Field	Description
TCPDRV359CD8	Unique identifier for this instance of TCP driver activity.
send	Indication that this event involves the TCP driver sending data.
2E8CD8	Address in memory of the data the TCP driver is sending.
(len 26)	Length of the data (in bytes).
queued	Indication that the TCP driver user process (in this case, remote source-route bridging) has transferred the data to the TCP driver to send.

The following line indicates that the TCP driver has sent the data that it had received from the TCP driver user, as shown in the previous line of output. The last field in the line (26) indicates that the 26 bytes of data were sent out as a single unit.

```
TCPDRV359CD8: output pak 2E8CD8 (len 26) (26)
```

The following line indicates that the TCP driver has received 42 bytes of data from the remote IP address. The TCP driver user (in this case, remote source-route bridging) has established an input threshold of 16 bytes for this connection. (The input threshold instructs the TCP driver to transfer data to the TCP driver user only when at least 16 bytes are present.)

```
TCPDRV359CD8: readf 42 bytes (Thresh 16)
```

debug ip tcp transactions

Use the **debug ip tcp transactions** EXEC command to display information on significant Transmission Control Protocol (TCP) transactions such as state changes, retransmissions, and duplicate packets. The **no** form of this command disables debugging output.

[no] debug ip tcp transactions

Usage Guidelines

This command is particularly useful for debugging a performance problem on a TCP/IP network that you have isolated above the data link layer.

The **debug ip tcp transactions** command displays output for packets the router sends and receives, but does not display output for packets it forwards.

Sample Display

Figure 2-115 shows sample **debug ip tcp transactions** output.

Figure 2-115 Sample Debug IP TCP Output

```
router# debug ip tcp transactions

TCP: sending SYN, seq 168108, ack 88655553
TCP0: Connection to 10.9.0.13:22530, advertising MSS 966
TCP0: state was LISTEN -> SYNRCVD [23 -> 10.9.0.13(22530)]
TCP0: state was SYNSENT -> SYNRCVD [23 -> 10.9.0.13(22530)]
TCP0: Connection to 10.9.0.13:22530, received MSS 956
TCP0: restart retransmission in 5996
TCP0: state was SYNRCVD -> ESTAB [23 -> 10.9.0.13(22530)]
TCP2: restart retransmission in 10689
TCP2: restart retransmission in 10641
TCP2: restart retransmission in 10633
TCP2: restart retransmission in 13384 -> 10.0.0.13(16151)]
TCP0: restart retransmission in 5996 [23 -> 10.0.0.13(16151)]
```

Table 2-60 describes significant fields shown in Figure 2-115.

Table 2-60 Debug IP TCP Field Descriptions

Field	Description
TCP:	Indicates that this is a TCP transaction.
sending SYN	Indicates that a synchronize packet is being sent.
seq 168108	Indicates the sequence number of the data being sent.
ack 88655553	Indicates the sequence number of the data being acknowledged.
TCP0:	Indicates the TTY number (0, in this case) with which this TCP connection is associated.
Connection to 10.9.0.13:22530	Indicates the remote address with which a connection has been established.
advertising MSS 966	Indicates the maximum segment size this side of the TCP connection is offering to the other side.

Table 2-60 Debug IP TCP Field Descriptions (Continued)

Field	Description
state was LISTEN -> SYNSENT	<p>Indicates that the TCP state machine changed state from LISTEN to SYNSENT. Possible TCP states follow:</p> <p>CLOSED—Connection closed.</p> <p>CLOSEWAIT—Received a FIN segment.</p> <p>CLOSING—Received a FIN/ACK segment.</p> <p>ESTAB—Connection established.</p> <p>FINWAIT 1—Sent a FIN segment to start closing the connection.</p> <p>FINWAIT 2—Waiting for a FIN segment.</p> <p>LASTACK—Sent a FIN segment in response to a received FIN segment.</p> <p>LISTEN—Listening for a connection request.</p> <p>SYNRCVD—Received a SYN segment, and responded.</p> <p>SYNSENT—Sent a SYN segment to start connection negotiation.</p> <p>TIMEWAIT—Waiting for network to clear segments for this connection before the network no longer recognizes the connection as valid. This must occur before a new connection can be set up.</p>
[23 -> 10.9.0.13(22530)]	<p>Within these brackets:</p> <p>The first field (23) indicates local TCP port.</p> <p>The second field (10.9.0.13) indicates the destination IP address.</p> <p>The third field (22530) indicates the destination TCP port.</p>
restart retransmission in 5996	<p>Indicates the number of milliseconds until the next retransmission takes place.</p>