Debug Commands

This chapter contains an alphabetical listing of the **debug** commands. 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 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 the **debug** commands are documented also varies. For example, for **debug** commands that generate lines of text, the output is described line by line. For **debug** commands that generate output in field format, tables are used to describe the fields.

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 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.

debug apple arp [*interface unit*]
no debug apple arp [*interface unit*]

Syntax Description

interface unit	(Optional) Information for a particular interface is to be displayed. For example, Ethernet0 specifies the first Ethernet interface; Ethernet1 specifies the second Ethernet interface. If you include this argument, you must specify both the interface type and unit number.

Command Mode

EXEC

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 2-1 shows 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)
```

S2723

Figure 2-1 Sample Debug Apple ARP Output

Explanations for representative lines of output in Figure 2-1 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 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.

debug apple errors [*interface unit*]
no debug apple errors [*interface unit*]

Syntax Description

interface unit	(Optional) Information for a particular interface is to be displayed. For example, Ethernet0 specifies the first Ethernet interface; Ethernet1 specifies the second Ethernet interface. If you include this argument, you must specify both the interface type and unit number.

Command Mode

EXEC

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 2-2 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.

```
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
```

S2651

Figure 2-2 Sample Debug Apple Errors Output

As Figure 2-2 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, RTMPRsp, 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 brrq, 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.

debug apple events [*interface unit*]
no debug apple events [*interface unit*]

Syntax Description

interface unit	(Optional) Information for a particular interface is to be displayed. For example, Ethernet0 specifies the first Ethernet interface; Ethernet1 specifies the second Ethernet interface. If you include this argument, you must specify both the interface type and unit number.

Command Mode

EXEC

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, however, the command generates numerous messages, they can indicate where the problems might lie.

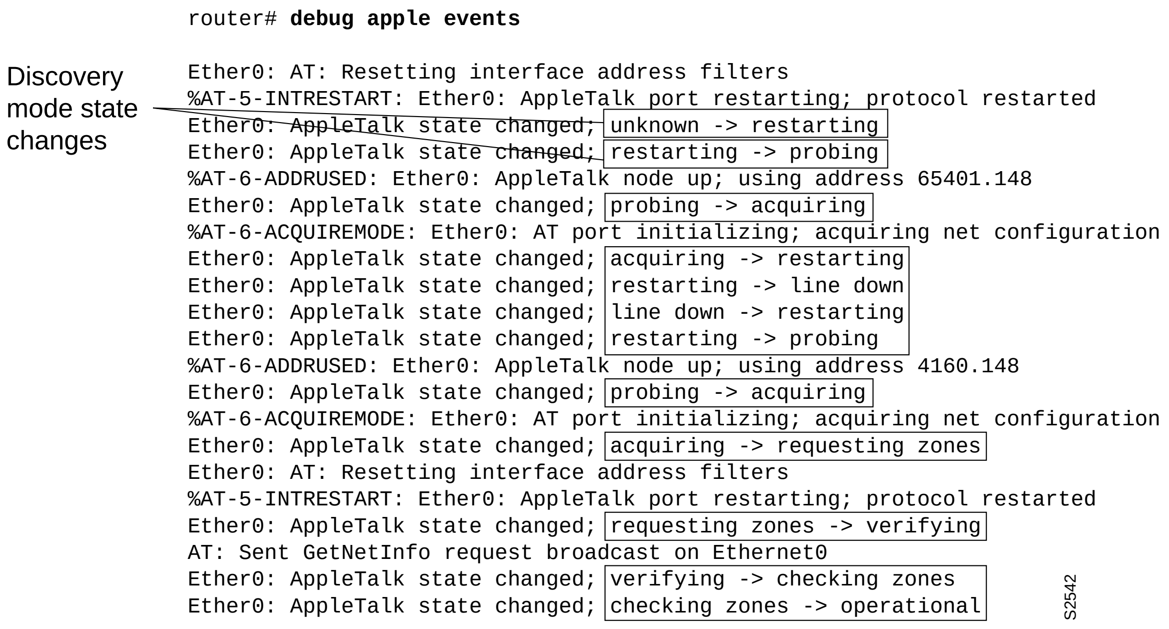
When configuring or making changes to a router or interface for AppleTalk, enable **debug apple events**. Doing so will alert you to the progress of the changes or to any errors that might result. Also use this command periodically when you suspect network problems.

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

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

Sample Display

Figure 2-3 shows sample **debug apple events** output that describes a nonseed router coming up in discovery mode.



```
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-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
```

Figure 2-3 Sample Debug Apple Events Output with Discovery Mode State Changes

As Figure 2-3 shows, the **debug apple events** command can be 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 2-3 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
```

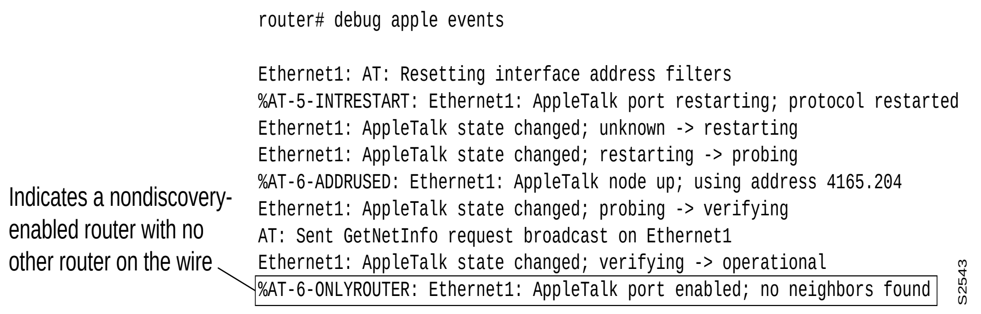
The following message indicates that the router is rechecking its list of zones for its cable:

```
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 2-4 shows sample **debug apple events** output that describes a nondiscovery-enabled router coming up when no other router is on the wire.



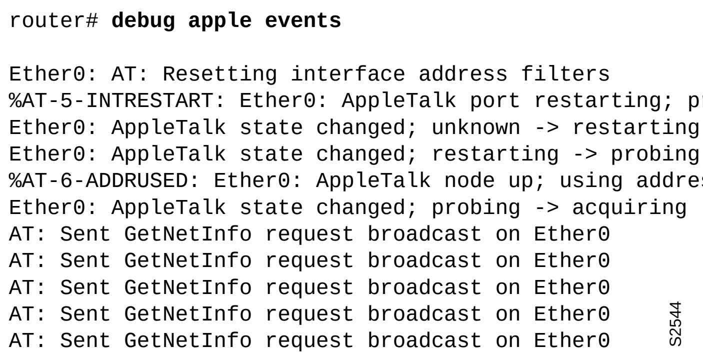
```
router# debug apple events

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
```

Indicates a nondiscovery-enabled router with no other router on the wire
S2543

Figure 2-4 Sample Debug Apple Events Output Showing Seed Coming Up by Itself

As Figure 2-4 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 2-4 indicates this situation.

Figure 2-5 shows sample **debug apple events** output that describes a discovery-enabled router coming up when there is no seed router on the wire.

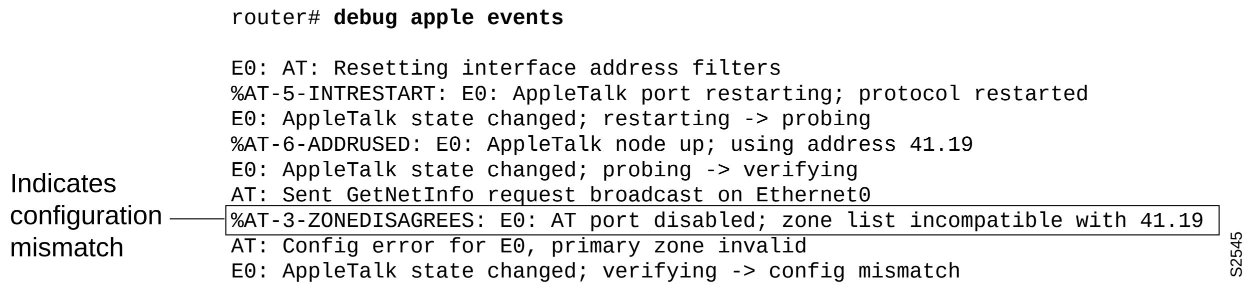


```
router# debug apple events

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
```

Figure 2-5 Sample Debug Apple Events Output Showing Nonseed with No Seed

As Figure 2-5 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 2-6 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.



```
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
```

Figure 2-6 Sample Debug Apple Events Output Showing Compatibility Conflict

The three configuration command lines that follow indicate the part of the router's configuration that caused the configuration mismatch shown in Figure 2-6:

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

The router shown in Figure 2-6 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 Marketign.

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.

debug apple nbp [*interface unit*]
no debug apple nbp [*interface unit*]

Syntax Description

interface unit	(Optional) Information for a particular interface is to be displayed. For example, Ethernet0 specifies the first Ethernet interface; Ethernet1 specifies the second Ethernet interface. If you include this argument, you must specify both the interface type and unit number.

Command Mode

EXEC

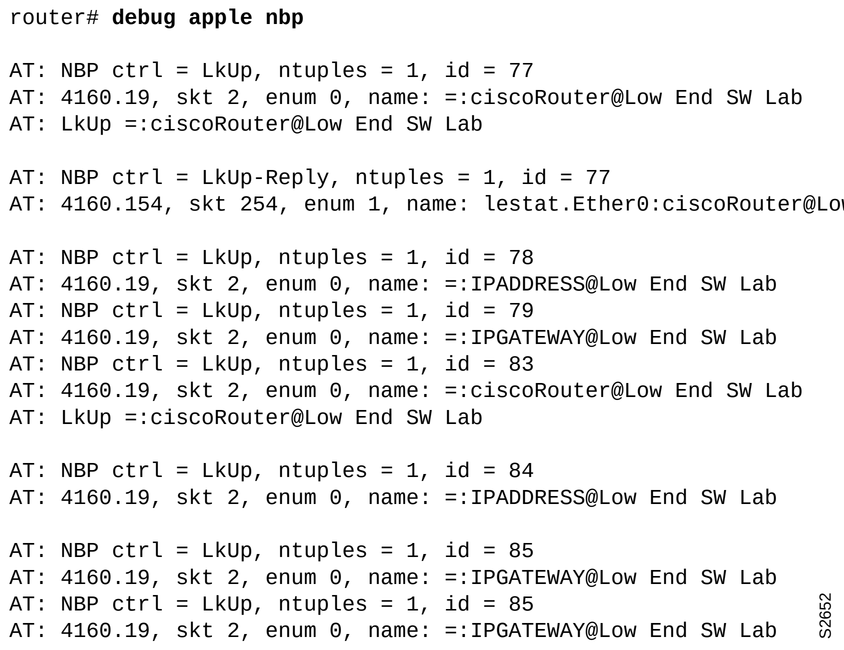
Usage Guidelines

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 2-7 shows 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
```

Figure 2-7 Sample Debug Apple NBP Output

The first three lines in Figure 2-7 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 2-1 describes the fields in the first line of output shown in Figure 2-7.

Table 2-1 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	Value that identifies the NBP lookup request.

Table 2-2 describes the fields in the second line of output shown in Figure 2-7.

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

Field	Description
AT:	Indicates that this message describes an AppleTalk packet.
4160.19	Network address of the requester.
skt 2	Internet socket address of the requester. The responder will send the NBP lookup reply to this socket address.
enum 0	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	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 2-7 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.

Since the router is defined as an object of type ciscoRouter in zone Low End SW Lab, it sends an NBP lookup reply in response to this NBP lookup request. The following two lines of output from Figure 2-7 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.

debug apple packet [*interface unit*]
no debug apple packet [*interface unit*]

Syntax Description

interface unit	(Optional) Information for a particular interface is to be displayed. For example, Ethernet0 specifies the first Ethernet interface; Ethernet1 specifies the second Ethernet interface. If you include this argument, you must specify both the interface type and unit number.

Command Mode

EXEC

Usage Guidelines

This command allows you to monitor the types of packets being slow switched. It will display at least one line of debugging output per AppleTalk packet processed.

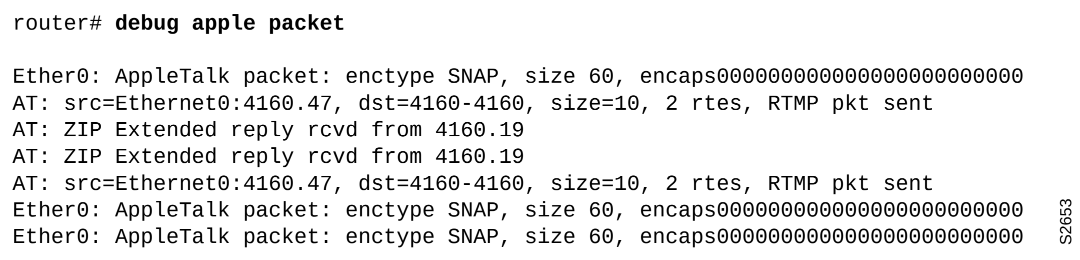
When invoked in conjunction with the **debug apple routing**, **debug apple zip**, and **debug apple nbp** commands, the **debug apple packet** command adds protocol processing information in addition to generic packet details. It also reports successful completion or failure information.

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 2-8 shows 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
```

Figure 2-8 Sample Debug Apple Packet Output

Table 2-3 describes the fields in the first line of output shown in Figure 2-8.

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

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

Table 2-4 describes the fields in the second line of output shown in Figure 2-8.

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

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

The third line in Figure 2-8 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 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.

debug apple routing [*interface unit*]
no debug apple routing [*interface unit*]

Syntax Description

[*interface unit*]	(Optional) Information for a particular interface is to be displayed. For example, Ethernet0 specifies the first Ethernet interface; Ethernet1 specifies the second Ethernet interface. If you include this argument, you must specify both the interface type and unit number.

Command Mode

EXEC

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 the router's CPU utilization is less than 50 percent.

Sample Display

Figure 2-9 shows 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
```

S2654

Figure 2-9 Sample Debug Apple Routing Output

Explanations for representative lines of the **debug apple routing** output in Figure 2-9 follow.

Table 2-5 describes the fields in the first line of sample **debug apple routing** output.

Table 2-5 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.
size = 19	Size of this RTMP packet (in bytes).
2 rtes	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 2-6 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 2-6 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	Number of routes in this RTMP update packet that the router did not already know about.
old 94	Number of routes in this RTMP update packet that the router already knew about.
bad 0	Number of routes the other router indicates have gone bad.
ign 0	Number of routes the other router indicates it does not care about.
dwn 0	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.

debug apple zip [*interface unit*]
no debug apple zip [*interface unit*]

Syntax Description

interface unit	(Optional) Information for a particular interface is to be displayed. For example, Ethernet0 specifies the first Ethernet interface; Ethernet1 specifies the second Ethernet interface. If you include this argument, you must specify both the interface type and unit number.

Command Mode

EXEC

Usage Guidelines

This command reports significant events such as 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.

The **debug apple zip** command can be used 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 2-10 shows 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-Orlando
```

S2655

Figure 2-10 Sample Debug Apple ZIP Output

Explanations of the lines of output shown in Figure 2-10 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-Orlando, and the zone length of that zone name is 10:

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

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.

debug arp
no debug arp

Syntax Description

This command has no arguments or keywords.

Command Mode

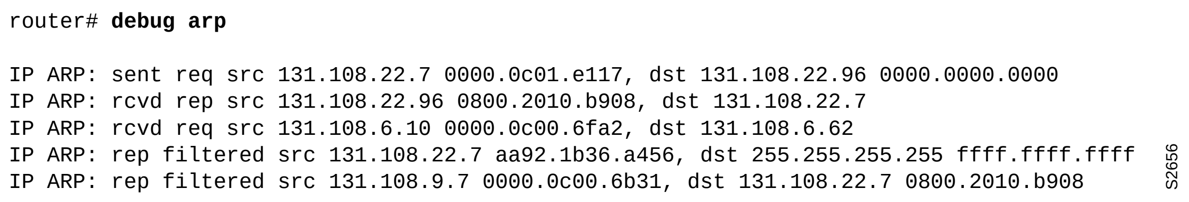
EXEC

Usage Guidelines

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

Sample Display

Figure 2-11 shows sample **debug arp** output.



```
router# debug arp

IP ARP: sent req src 131.108.22.7 0000.0c01.e117, dst 131.108.22.96 0000.0000.0000
IP ARP: rcvd rep src 131.108.22.96 0800.2010.b908, dst 131.108.22.7
IP ARP: rcvd req src 131.108.6.10 0000.0c00.6fa2, dst 131.108.6.62
IP ARP: rep filtered src 131.108.22.7 aa92.1b36.a456, dst 255.255.255.255 ffff.ffff.ffff
IP ARP: rep filtered src 131.108.9.7 0000.0c00.6b31, dst 131.108.22.7 0800.2010.b908
```

Figure 2-11 Sample Debug ARP Output

In Figure 2-11, 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 131.108.22.7 and MAC address 0000.0c01.e117 sent an ARP request for the MAC address of the host at 131.108.22.96. 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 131.108.22.7 0000.0c01.e117, dst 131.108.22.96 \
0000.0000.0000
```

The second line indicates that the router at IP address 131.108.22.7 receives a reply from the host at 131.108.22.96 indicating that its MAC address is 0800.2010.b908:

```
IP ARP: rcvd rep src 131.108.22.96 0800.2010.b908, dst 131.108.22.7
```

The third line indicates that the router receives an ARP request from the host at 131.108.6.10 requesting the MAC address for the host at 131.108.6.62:

```
IP ARP: rcvd req src 131.108.6.10 0000.0c00.6fa2, dst 131.108.6.62
```

The fourth line indicates that another host on the network attempted to send the router an ARP reply for the router's own address. The router ignores such bogus replies. Usually, this can happen if someone is running a bridge in parallel with the router and is allowing ARP to be bridged. It indicates a network misconfiguration.

```
IP ARP: rep filtered src 131.108.22.7 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 131.108.9.7, but the router does not know that that 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 131.108.9.7 0000.0c00.6b31, dst 131.108.22.7 \
0800.2010.b908
```

debug atm errors

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

debug atm errors
no debug atm errors

Syntax Description

This command has no arguments or keywords.

Command Mode

EXEC

Sample Display

Figure 2-12 shows sample **debug atm errors** output.

```
router# debug atm errors
ATM(ATM2/0): Encapsulation error, link=7, host=836CA86D.
```

S2841

Figure 2-12 Sample Debug ATM Errors Output

The line of output in Figure 2-12 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.

debug atm events

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

debug atm events
no debug atm events

Syntax Description

This command has no arguments or keywords.

Command Mode

EXEC

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, they can indicate where the problems might lie.

When configuring or making changes to a router or interface for ATM, enable **debug atm events**. Doing so will alert 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 2-13 shows 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
```

S2840

Figure 2-13 Sample Debug ATM Events Output

Table 2-7 describes significant fields in the output shown in Figure 2-13.

Table 2-7 Debug ATM Events Field Descriptions

Field	Description
PLIM type	Indicates the interface rate in Mbps. Possible values are 1 = TAXI(4B5B) 100 Mbps 2 = SONET 155 Mbps 3 = E3 34 Mbps
state	Indicates current state of the AIP. Possible values are 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
	0x0800 = AIP crashed, reload may be required
	0x0400 = AIP detected a carrier state change
	0x0n00 = Command completion status. Command completion status codes are
	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 2-13 follow.

The following line indicates that the 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 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 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.

debug atm packet
no debug atm packet

Syntax Description

This command has no arguments or keywords.

Command Mode

EXEC

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 correctly or are being 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 has 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 users on the system will not be adversely affected.

Sample Display

Figure 2-14 shows sample **debug atm packet** output.

```
router# debug atm packets
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
```

S2842

Figure 2-14 Sample Debug ATM Packet Output

Table 2-8 describes significant fields shown in Figure 2-14.

Table 2-8 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: 0x*n*	Indicates the virtual circuit associated with this packet, where *n* is some value.
DM: 0x*nnnn*	Indicates the descriptor mode bits on output only, where *nnnn* is a hexadecimal value.
ETYPE:	Ethernet type for this packet.
Length	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 DCFF 0105
```

debug broadcast

Use the **debug broadcast** EXEC command to display information on MAC broadcast packets. The **no** form of this command disables debugging output.

debug broadcast
no debug broadcast

Syntax Description

This command has no arguments or keywords.

Command Mode

EXEC

Usage Guidelines

Depending on the type of interface and the type of encapsulation used on that interface, the **debug broadcast** command can produce a wide range of messages.

Sample Display

Figure 2-15 shows sample **debug broadcast** output. Notice how similar it is to the **debug packet** output.

```
router# debug broadcast

Ethernet0: Broadcast ARPA, src 0000.0c00.6fa4, dst ffff.ffff.ffff, type 0x0800,
data 4500002800000000FF11EA7B, len 60
Serial3: Broadcast HDLC, size 64, type 0x800, flags 0x8F00
Serial2: Broadcast PPP, size 128
Serial7: Broadcast FRAME-RELAY, size 174, type 0x800, DLCI 7a
```

S2657

Figure 2-15 Sample Debug Broadcast Output

Table 2-9 describes significant fields shown in Figure 2-15.

Table 2-9 Debug Broadcast Field Descriptions

Field	Description
Ethernet0	Name of Ethernet interface that received the packet.
Broadcast	States that this packet was a broadcast packet.
ARPA	States that this packet uses ARPA-style encapsulation. Possible encapsulation styles vary depending on the media command mode (MCM) and encapsulation style, as follows: **Ethernet (MCM)** *Encapsulation Style* APOLLO ARP ETHERTALK ISO1 ISO3 LLC2 NOVELL-ETHER SNAP
	FDDI (MCM) *Encapsulation Style* APOLLO ISO1 ISO3 LLC2 SNAP
	Serial (MCM) *Encapsulation Style* BFEX25 BRIDGE DDN-X25 DDNX25-DCE ETHERTALK FRAME-RELAY HDLC HDH LAPB LAPBDCE MULTI-LAPB PPP SDLC-PRIMARY SDLC-SECONDARY SLIP SMDS STUN X25 X25-DCE

Field	Description
	Token Ring (MCM) *Encapsulation Style* 3COM-TR ISO1 ISO3 MAC LLC2 NOVELL-TR SNAP VINES-TR
src 0000.0c00.6fa4	MAC address of the node generating the packet.
dst ffff.ffff.ffff.ffff	MAC address of the destination node for the packet. This address is always the MAC broadcast address.
type 0x0800	Packet type (IP in this case).
data ...	First 12 bytes of the datagram following the MAC header.
len 60	Length of the message that the interface received from the wire (in bytes).
size 128	Length of the message that the interface received from the wire (in bytes).
flags 0x8F00	HDLC or PPP flags field.
DLCI 7a	The DLCI number on Frame Relay.

debug clns esis events

Use the **debug clns esis events** EXEC command to displays uncommon 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.

debug clns esis events
no debug clns esis events

Syntax Description

This command has no arguments or keywords.

Command Mode

EXEC

Sample Display

Figure 2-16 shows 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
```

Figure 2-16 Sample Debug CLNS ESIS Events Output

Explanations for individual lines of output from Figure 2-16 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 router's NET address is 49.0001.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 ES-IS packets that the router has received and sent. The **no** form of this command disables debugging output.

debug clns esis packets
no debug clns esis packets

Syntax Description

This command has no arguments or keywords.

Command Mode

EXEC

Sample Display

Figure 2-17 shows 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.O906.4023.00, HT 299, HLEN 34
IS-IS: ESH from 0000.0c00.bda8 (Ethernet0), HT 300
```

Figure 2-17 Sample Debug CLNS ESIS Packets Output

Explanations for individual lines of output from Figure 2-17 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 router's NET 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 router's NET 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 (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 router's NET 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 (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.

debug clns events
no debug clns events

Syntax Description

This command has no arguments or keywords.

Command Mode

EXEC

Sample Display

Figure 2-18 shows 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
```

S2660

Figure 2-18 Sample Debug CLNS Events Output

Explanations for individual lines of output from Figure 2-18 follow.

The following line indicates that the router received an echo PDU on Ethernet3 from source 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.

debug clns igrp packets
no debug clns igrp packets

Syntax Description

This command has no arguments or keywords.

Command Mode

EXEC

Sample Display

Figure 2-19 shows 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    S2661
```

Figure 2-19 Sample Debug CLNS IGRP Packets Output

Explanations for individual lines of output from Figure 2-19 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 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.

debug clns packet
no debug clns packet

Syntax Description

This command has no arguments or keywords.

Command Mode

EXEC

Sample Display

Figure 2-20 shows 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)
```

S2662

Figure 2-20 Sample Debug CLNS Packet Output

Explanations for individual lines of output from Figure 2-20 follow.

In the following lines, the first line indicates that a CLNS packet of size 157 bytes is being forwarded. The second line indicates the 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 of all CLNS routing cache updates and activities involving the CLNS routing table. The **no** form of this command disables debugging output.

debug clns routing
no debug clns routing

Syntax Description

This command has no arguments or keywords.

Command Mode

EXEC

Sample Display

Figure 2-21 shows 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
```

Figure 2-21 Sample Debug CLNS Routing Output

Explanations for individual lines of output from Figure 2-21 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 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 compress

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

debug compress
no debug compress

Syntax Description

This command has no arguments or keywords.

Command Mode

EXEC

Sample Display

Figure 2-22 shows 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
```

Figure 2-22 Sample Debug Compress Output

Table 2-10 describes significant fields shown in Figure 2-22.

Table 2-10 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 LAPB reset will occur. On LAPB reset, a compression reinitialization occurs. Compression reinitialization initializes the dictionaries and xmt_paks and rcv_sync counts.

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.

debug decnet adj
no debug decnet adj

Syntax Description

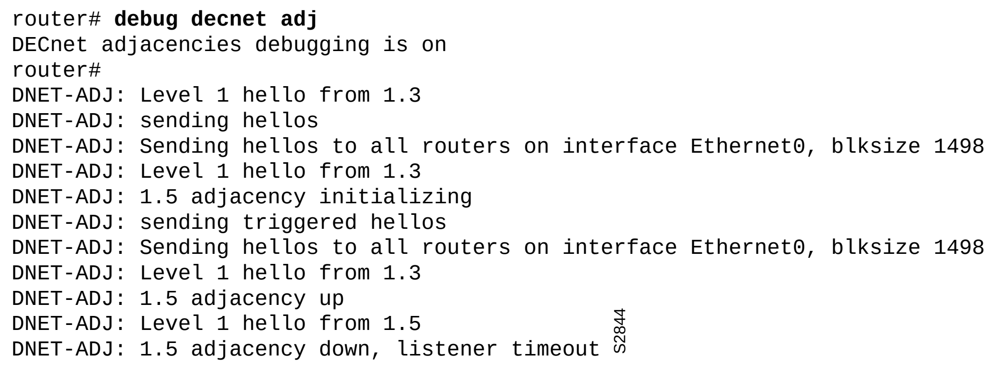
This command has no arguments or keywords.

Command Mode

EXEC

Sample Display

Figure 2-23 shows 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
```

Figure 2-23 Sample Debug DECnet Adj Output

Explanations for representative lines of output in Figure 2-23 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 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.

debug decnet connects
no debug decnet connects

Syntax Description

This command has no arguments or keywords.

Command Mode

EXEC

Usage Guidelines

When using 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
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.

Sample Display

Figure 2-24 shows 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"
```

S2664

Figure 2-24 Sample Debug DECnet Connects Output

Table 2-11 describes significant fields shown in Figure 2-24.

Table 2-11 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.
dstobj = 42	Indicates that DECnet object 42 is the destination.
id="USER"	Indicates the access user.

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 your router's configuration.

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.

debug decnet events
no debug decnet events

Syntax Description

This command has no arguments or keywords.

Command Mode

EXEC

Sample Display

Figure 2-25 shows sample **debug decnet events** output.

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

Figure 2-25 Sample Debug DECnet Events Output

Explanations for representative lines of output in Figure 2-25 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.

debug decnet packet
no debug decnet packet

Syntax Description

This command has no arguments or keywords.

Command Mode

EXEC

Sample Display

Figure 2-26 shows 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
```

S2846

Figure 2-26 Sample Debug DECnet Packet Output

Explanations for individual lines of output from Figure 2-26 follow.

The following line indicates that the router is sending a converted packet addressed to node 1.10 to Phase V:

```
DNET-PKT: src 1.3 dst 1.10 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 SNPA (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.

debug decnet routing
no debug decnet routing

Syntax Description

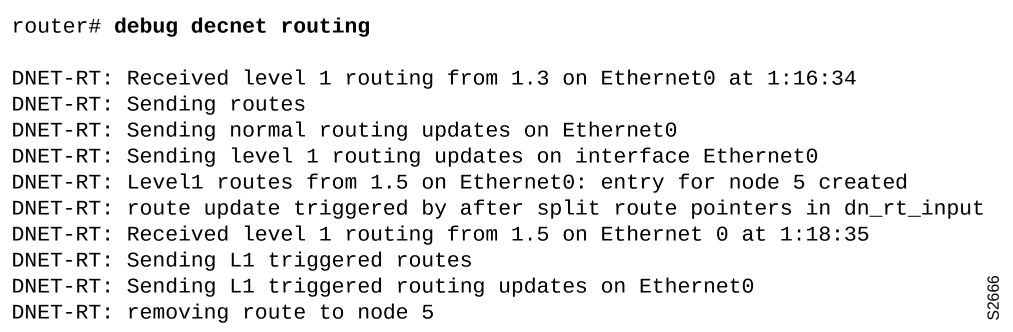
This command has no arguments or keywords.

Command Mode

EXEC

Sample Display

Figure 2-27 shows 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
```

Figure 2-27 Sample Debug DECnet Routing Output

Explanations for individual lines of output from Figure 2-27 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 1.5 because the adjacency with node 1.5 timed out, or the route to node 1.5 through a next-hop router went away:

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

debug dialer

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

debug dialer
no debug dialer

Syntax Description

This command has no arguments or keywords.

Command Mode

EXEC

Usage Guidelines

Table 2-12 describes the error messages that the **debug dialer** command can generate for a serial interface being used as a V.25bis dialer for dial-on-demand routing (DDR).

Table 2-12 Debug Dialer Message Descriptions for DDR

Message	Description
Serial 0: Dialer result = *xxxxxxxxxx*	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 *xxxxxxxxxx* 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 *xxxxxxxxxx*	This message indicates that a packet has been received that passes the dial-on-demand access lists. That packet causes dialing of a phone number. The *xxxxxxxxxx* variable is the number being called.
Serial 0: Unable to dial *xxxxxxxxxx*	This message is displayed if for some reason, the phone call could not be placed. This 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 attempts to hang 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 their corresponding dialer timer expires. They are mostly informational, but are useful when debugging a disconnected call or call failure.

When DDR is enabled on the interface, information concerning the cause of any calls (called Dialing cause) may be 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=131.108.1.111 d=131.108.2.22)
```

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 *Router Products Command Reference* publication.

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

debug frame-relay

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

debug frame-relay
no debug frame-relay

Syntax Description

This command has no arguments or keywords.

Command Mode

EXEC

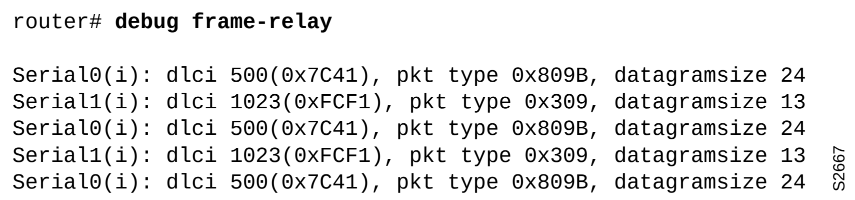
Usage Guidelines

This command helps you to 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 packets** command.

Sample Display

Figure 2-28 shows 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
```

Figure 2-28 Sample Debug Frame-Relay Output

Table 2-13 describes significant fields shown in Figure 2-28.

Table 2-13 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)	Value of the DLCI for this packet in decimal (and q922). In this case, 500 has been configured as the multicast DLCI.

Field	Description
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
	Possible supported Ethernet type codes follow:
	0x0201—IP on 3MB net
	0x0201—Xerox ARP on 10MB nets
	0xCC—RFC 1294 (only for IP)
	0x0600—XNS
	0x0800—IP on 10 MB net
	0x0806—IP ARP
	0x0808—Frame Relay ARP
	0x0BAD—VINES IP
	0x0BAE—VINES loopback protocol
	0x0BAF—VINES Echo
	0x6001—DEC MOP booting protocol
	0x6002—DEC MOP console protocol
	0x6003—DECnet Phase IV on Ethernet
	0x6004—DEC LAT on Ethernet
	0x8005—HP Probe
	0x8035—RARP
	0x8038—DEC spanning tree
	0x809b—Apple EtherTalk
	0x80f3—AppleTalk ARP
	0x8019—Apollo domain
	0x80C4—VINES IP
	0x80C5— VINES ECHO
	0x8137—IPX
	0x9000—Ethernet loopback packet IP

Field	Description
pkt type 0x809B (continued)	Possible HDLC type codes follow:
	0x1A58— IPX, standard form
	0xFEFE—CLNS
	0xEFEF—ES-IS
	0x1998—Uncompressed TCP
	0x1999—Compressed TCP
	0x6558—Serial line bridging
datagramsize 24	Size of this datagram (in bytes)

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.

debug frame-relay events
no debug frame-relay events

Syntax Description

This command has no arguments or keywords.

Command Mode

EXEC

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 2-29 shows sample **debug frame-relay events** output.

```
router# debug frame-relay events

Serial2(i): reply rcvd 131.108.170.26 126
Serial2(i): reply rcvd 131.108.170.28 128
Serial2(i): reply rcvd 131.108.170.34 134
Serial2(i): reply rcvd 131.108.170.38 144
Serial2(i): reply rcvd 131.108.170.41 228
Serial2(i): reply rcvd 131.108.170.65 325
```

S2668

Figure 2-29 Sample Debug Frame-Relay Events Output

As Figure 2-29 shows, **debug frame-relay events** returns one specific message type. The first line, for example, indicates that IP address 131.108.170.26 sent a Frame Relay ARP reply; this packet was received as input on the Serial2 interface. The last field (126) is the DLCI to use when communicating with the responding router.

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.

debug frame-relay lmi
no debug frame-relay lmi

Syntax Description

This command has no arguments or keywords.

Command Mode

EXEC

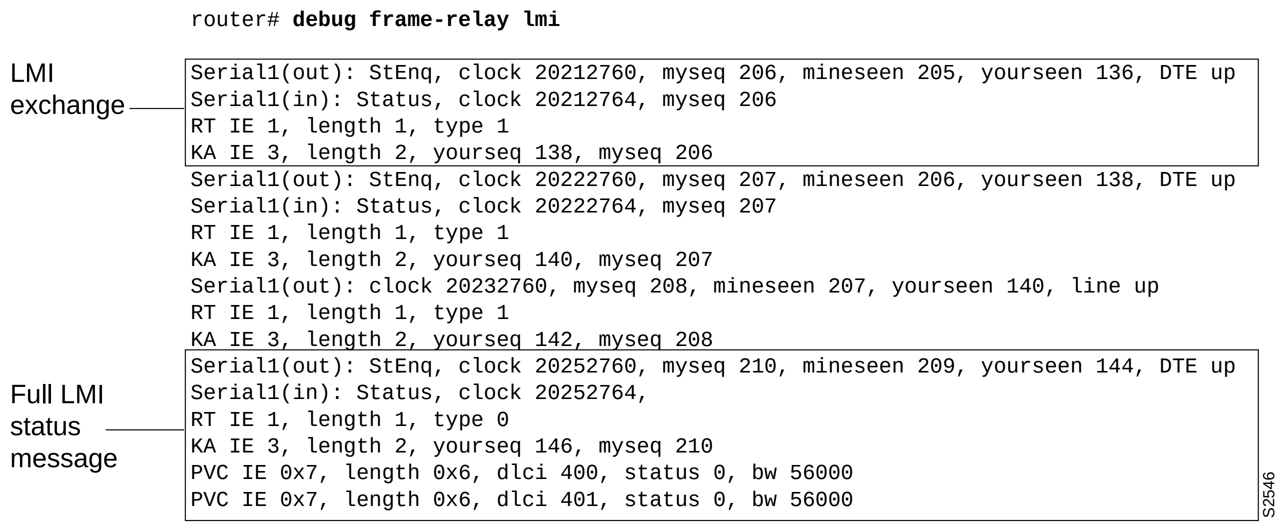
Usage Guidelines

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 2-30 shows sample **debug frame-relay lmi** output.



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

Figure 2-30 Sample Debug Frame-Relay LMI Output

In Figure 2-30, 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 2-30 comprise a full LMI status message that includes a description of the router's two permanent virtual circuits (PVCs).

Table 2-14 describes significant fields in the first line of the **debug frame-relay lmi** output shown in Figure 2-30.

Table 2-14 Debug Frame-Relay LMI Field Descriptions—Part 1

Field	Description
Serial1(out)	Indicates that the LMI request was sent out on the Serial1 interface.
StEnq	Command Mode of message: StEnq—Status Enquiry 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	Indicates the line protocol up/down state for the DTE (user) port.

Table 2-15 describes significant fields in the third and fourth lines of **debug frame-relay lmi** output shown in Figure 2-30.

Table 2-15 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 2-16 describes significant fields in the last line of **debug frame-relay lmi** output shown in Figure 2-30.

Table 2-16 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).
dlci 401	DLCI decimal value for this PVC.
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 packets

Use the **debug frame-relay packets** 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.

debug frame-relay packets
no debug frame-relay packets

Syntax Description

This command has no arguments or keywords.

Command Mode

EXEC

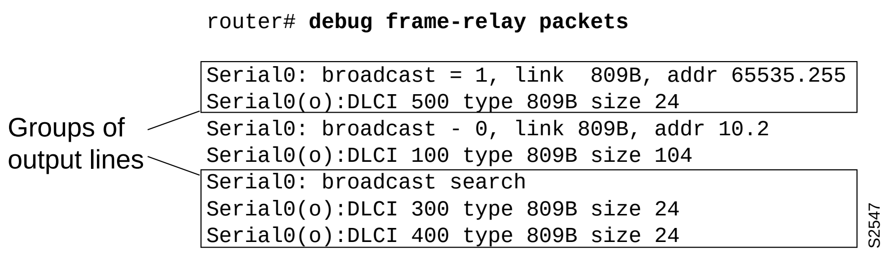
Usage Guidelines

This command helps you to analyze the packets that have been sent on a Frame Relay interface. Because the **debug frame-relay packets** command generates large amounts 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 *received* on a Frame Relay interface, use the **debug frame-relay** command.

Sample Display

Figure 2-31 shows sample **debug frame-relay packets** output.



```
router# debug frame-relay packets

Serial0: broadcast = 1, link  809B, addr 65535.255
Serial0(o):DLCI 500 type 809B size 24
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
```

Figure 2-31 Sample Debug Frame-Relay Packets Output

As Figure 2-31 shows, **debug frame-relay packets** output comprises 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 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 2-31 describe a single Frame Relay packet that was sent out on two DLCIs.

Table 2-17 describes significant fields shown in the first pair of output lines in Figure 2-31.

Table 2-17 Debug Frame-Relay Packets Field Descriptions

Field	Description
Serial0:	Indicates the interface that has sent the Frame Relay packet.

broadcast = 1	Indicates the 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	Indicates the packet type, as documented under "debug frame relay."
addr 65535.255	Indicates the 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	Indicates the 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 2-31 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 2-31. They describe a Frame Relay packet sent to a true broadcast address.

```
Serial1: broadcast search
Serial1(o):DLCI 400 type 800 size 288
```

debug ip icmp

Use the **debug ip icmp** EXEC command to display information on ICMP transactions. The **no** form of this command disables debugging output.

debug ip icmp
no debug ip icmp

Syntax Description

This command has no arguments or keywords.

Command Mode

EXEC

Usage Guidelines

This command is useful for determining whether the router is sending and/or receiving ICMP messages; for example, when troubleshooting an end-to-end connection problem.

Sample Display

Figure 2-32 shows sample **debug ip icmp** output.

```
router# debug ip icmp

ICMP: rcvd type 3, code 1, from 128.95.192.4
ICMP: src 36.56.0.202, dst 131.108.16.1, echo reply
ICMP: dst (131.120.1.0) port unreachable rcv from 131.120.1.15
ICMP: src 131.108.12.35, dst 131.108.20.7, echo reply
ICMP: dst (255.255.255.255) protocol unreachable rcv from 192.31.7.21
ICMP: dst (131.120.1.0) port unreachable rcv from 131.120.1.15
ICMP: dst (255.255.255.255) protocol unreachable rcv from 192.31.7.21
ICMP: dst (131.120.1.0) port unreachable rcv from 131.120.1.15
ICMP: src 36.56.0.202, dst 131.108.16.1, echo reply
ICMP: dst (131.120.1.0) port unreachable rcv from 131.120.1.15
ICMP: dst (255.255.255.255) protocol unreachable rcv from 192.31.7.21
ICMP: dst (131.120.1.0) port unreachable rcv from 131.120.1.15
```

S2669

Figure 2-32 Sample Debug IP ICMP Output

Table 2-18 describes significant fields shown in the first line of **debug ip icmp** output shown in Figure 2-32.

Table 2-18 Debug IP ICMP Field Descriptions—Part 1

Field	Description
ICMP:	Indicates that this message describes an ICMP packet.
rcvd type 3	The type field can be one of the following: 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	This field is a code. The meaning of the code depends upon the type field value: Echo and Echo Reply—The code field is always zero. Destination Unreachable—The code field can have the following values: 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: 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.

Field	Description
code 1 (continued)	Time Exceeded—The code field can have the following values:
	0—Time to live exceeded in transit
	1—Fragment reassembly time exceeded
	Parameter Problem—The code field can have the following values:
	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 128.95.192.4	Indicates the source address of the ICMP packet.

Table 2-19 describes significant fields shown in the second line of **debug ip icmp** output in Figure 2-32.

Table 2-19 Debug IP ICMP Field Descriptions—Part 2

Field	Description
ICMP:	Indicates that this message describes an ICMP packet.
src 36.56.0.202	The address of the sender of the echo.
dst 131.108.16.1	The address of the receiving router.
echo reply	Indicates 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 160.89.80.23 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 160.89.80.31
ICMP: unexpected mask reply 255.255.255.0 from 160.89.80.32
```

The following output indicates that the router sent a redirect packet to the host at address 160.89.80.31, instructing that host to use the gateway at address 160.89.80.23 in order to reach the host at destination address 131.108.1.111:

```
ICMP: redirect sent to 160.89.80.31 for dest 131.108.1.111 use gw 160.89.80.23
```

The following message indicates that the router received a redirect packet from the host at address 160.89.80.23, instructing the router to use the gateway at address 160.89.80.28 in order to reach the host at destination address 160.89.81.34:

```
ICMP: redirect rcvd from 160.89.80.23 -- for 160.89.81.34 use gw 160.89.80.28
```

The following message is displayed when the router sends an ICMP packet to the source address (160.89.94.31 in this case), indicating that the destination address (131.108.13.33 in this case) is unreachable:

```
ICMP: dst (131.108.13.33) host unreachable sent to 160.89.94.31
```

The following message is displayed when the router receives an ICMP packet from an intermediate address (160.89.98.32 in this case), indicating that the destination address (131.108.13.33 in this case) is unreachable:

```
ICMP: dst (131.108.13.33) host unreachable rcv from 160.89.98.32
```

Depending on the code received (as Table 2-18 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 128.95.1.4 (dest was 131.108.1.111)
```

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 131.108.1.111)
```

Based on the preceding information, the remaining output can be easily understood.

```
ICMP: parameter problem rcvd 160.89.80.32
ICMP: source quench rcvd 160.89.80.32
ICMP: source quench sent to 128.121.1.50 (dest was 131.108.1.111)
ICMP: sending time stamp reply to 160.89.80.45
ICMP: sending info reply to 160.89.80.12
ICMP: rdp advert rcvd type 9, code 0, from 160.89.80.23
ICMP: rdp solicit rcvd type 10, code 0, from 160.89.80.43
```

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."

debug ip igrp events

Use the **debug ip igrp events** EXEC command to display information of IGRP routing messages that indicate 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.

debug ip igrp events [*ip-address*]
no debug ip igrp events [*ip-address*]

Syntax Description

ip-address	(Optional) IP address of an IGRP neighbor

Command Mode

EXEC

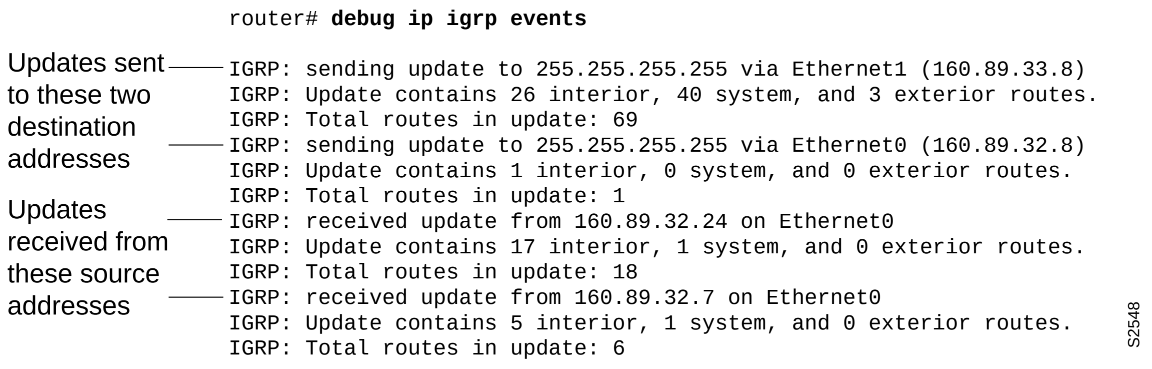
Usage Guidelines

If the IP address of an IGRP neighbor is specified, the resulting **debug ip igrp events** output will include 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 2-33 shows sample **debug ip igrp events** output.


Updates sent to these two destination addresses
Updates received from these source addresses
S2548

```
router# debug ip igrp events
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
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
```

Figure 2-33 Sample Debug IP IGRP Events Output

Figure 2-33 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. Explanations for representative lines of output from Figure 2-33 follow.

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 information on IGRP routing transactions. The **no** form of this command disables debugging output.

debug ip igrp transaction [*ip-address*]
no debug ip igrp transaction [*ip-address*]

Syntax Description

ip-address	(Optional) IP address of an IGRP neighbor

Command Mode

EXEC

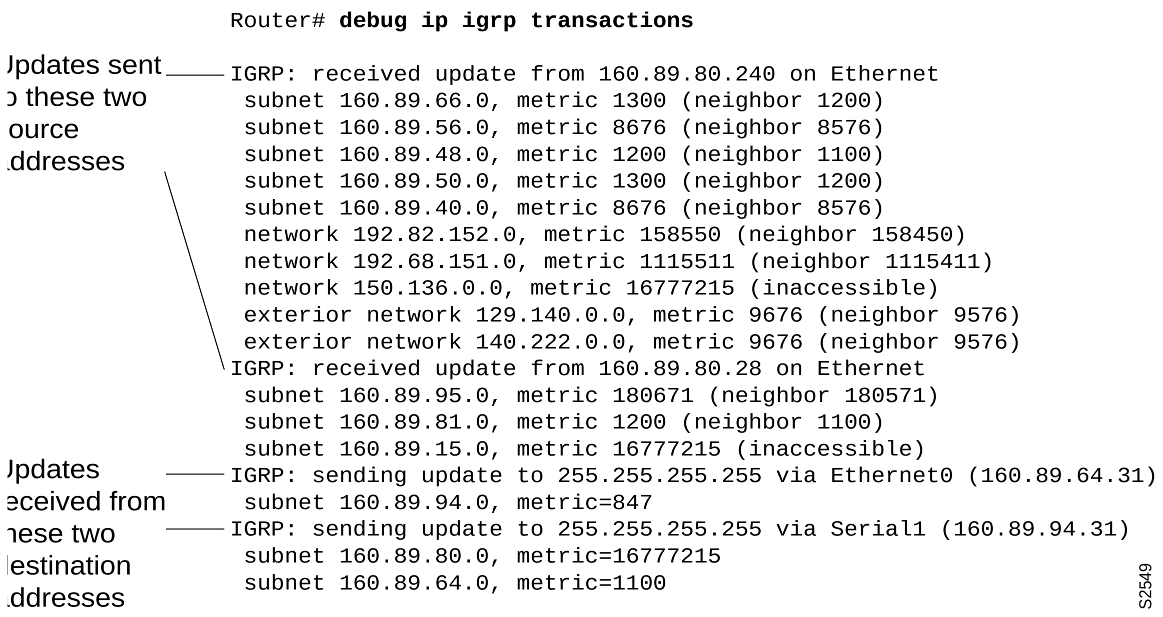
Usage Guidelines

If the IP address of an IGRP neighbor is specified, the resulting **debug ip igrp transaction** output will include 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 2-34 shows sample **debug ip igrp transaction** output.



```
Router# debug ip igrp transactions

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)
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 2-34 Sample Debug IP IGRP Transaction Output

Figure 2-34 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 2-34 is self-explanatory.

On the second line in Figure 2-34, 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 2-34 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 160.89.64.43
```

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 160.89.64.234, should be system 109
```

debug ip ospf events

Use the **debug ip ospf events** EXEC command to display information on OSPF-related events, such as adjacencies, flooding information, designated router selection, and SPF calculation. The **no** form of this command disables debugging output.

debug ip ospf events
no debug ip ospf events

Syntax Description

This command has no arguments or keywords.

Command Mode

EXEC

Sample Display

Figure 2-35 shows 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
```

S2670

Figure 2-35 Sample Debug IP OSPF Events Output

The **debug ip ospf events** output shown in Figure 2-35 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 transit area and the other is a part of a stub area, as explained in RFC 1247).

```
OSPF: hello packet with mismatched E bit
```

debug ip packet

Use the **debug ip packet** EXEC command to display general IP debugging information and IPSO security transactions. The **no** form of this command disables debugging output.

debug ip packet [*access-list-number*]
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.

Command Mode

EXEC

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 users on the system will not be adversely affected.

Sample Display

Figure 2-36 shows sample **debug ip packet** output.

```
router# debug ip packet

IP: s=131.108.13.44 (Fddi0), d=157.125.254.1 (Serial2), g=131.108.16.2, forward
IP: s=131.108.1.57 (Ethernet4), d=192.36.125.2 (Serial2), g=131.108.16.2, forward
IP: s=131.108.1.6 (Ethernet4), d=255.255.255.255, rcvd 2
IP: s=131.108.1.55 (Ethernet4), d=131.108.2.42 (Fddi0), g=131.108.13.6, forward
IP: s=131.108.89.33 (Ethernet2), d=131.130.2.156 (Serial2), g=131.108.16.2, forward
IP: s=131.108.1.27 (Ethernet4), d=131.108.43.126 (Fddi1), g=131.108.23.5, forward
IP: s=131.108.1.27 (Ethernet4), d=131.108.43.126 (Fddi0), g=131.108.13.6, forward
IP: s=131.108.20.32 (Ethernet2), d=255.255.255.255, rcvd 2
IP: s=131.108.1.57 (Ethernet4), d=192.36.125.2 (Serial2), g=131.108.16.2, access denied
```

S2671

Figure 2-36 Sample Debug IP Packet Output

Figure 2-36 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 2-20 describes the fields shown in the first line of Figure 2-36.

Table 2-20 Debug IP Packet Field Descriptions

Field	Description
IP:	Indicates that this is an IP packet.
s = 131.108.13.44 (Fddi0)	Indicates the source address of the packet and the name of the interface that received the packet.
d = 157.125.254.1 (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 = 131.108.16.2	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 2-36.

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 2-21 describes.

Table 2-21 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. These reasons can be any of the following:

- no basic
- no basic, no resp
- reserved class
- reserved class, no resp
- class too low, no resp
- class too high
- class too high, bad auths, no resp
- unrecognized class
- unrecognized class, no resp
- multiple basic

- multiple basic, no resp
- auth too low, no resp
- auth too high
- compartment bits not dominated by max sensitivity level
- compartment bits don't dominate min 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 rip

Use the **debug ip rip** EXEC command to display information on RIP routing transactions. The **no** form of this command disables debugging output.

debug ip rip
no debug ip rip

Syntax Description

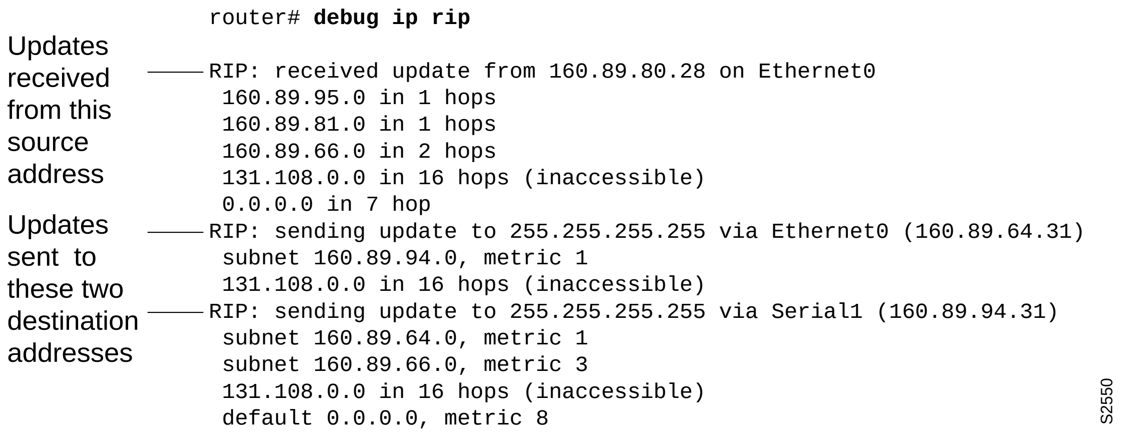
This command has no arguments or keywords.

Command Mode

EXEC

Sample Display

Figure 2-37 shows sample **debug ip rip** output.


Updates received from this source address
Updates sent to these two destination addresses
S2550

```
router# debug ip rip

RIP: received update from 160.89.80.28 on Ethernet0
 160.89.95.0 in 1 hops
 160.89.81.0 in 1 hops
 160.89.66.0 in 2 hops
 131.108.0.0 in 16 hops (inaccessible)
 0.0.0.0 in 7 hop
RIP: sending update to 255.255.255.255 via Ethernet0 (160.89.64.31)
 subnet 160.89.94.0, metric 1
 131.108.0.0 in 16 hops (inaccessible)
RIP: sending update to 255.255.255.255 via Serial1 (160.89.94.31)
 subnet 160.89.64.0, metric 1
 subnet 160.89.66.0, metric 3
 131.108.0.0 in 16 hops (inaccessible)
 default 0.0.0.0, metric 8
```

Figure 2-37 Sample Debug IP RIP Output

Figure 2-37 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 2-37 is self-explanatory.

The second line in Figure 2-37 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 2-37 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 some event occurs such as an interface transitioning or the 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 RIP routing table updates and route-cache updates. The **no** form of this command disables debugging output.

debug ip routing
no debug ip routing

Syntax Description

This command has no arguments or keywords.

Command Mode

EXEC

Sample Display

Figure 2-38 shows sample **debug ip routing** output.

```
router# debug ip routing
ip routing debugging is on
RT: add 198.93.168.0 255.255.255.0 via 198.92.76.30, igrp metric [100/3020]
RT: metric change to 198.93.168.0 via 198.92.76.30, igrp metric [100/3020]
        new metric [100/2930]
IP: cache invalidation from 0x115248 0x1378A, new version 5736
RT: add 198.133.219.0 255.255.255.0 via 198.92.76.30, igrp metric [100/16200]
RT: metric change to 198.133.219.0 via 198.92.76.30, igrp metric [100/16200]
        new metric [100/10816]
RT: delete route to 198.133.219.0 via 198.92.76.30, igrp metric [100/10816]
RT: no routes to 198.133.219.0, entering holddown
IP: cache invalidation from 0x115248 0x1378A, new version 5737
RT: 198.133.219.0 came out of holddown
RT: garbage collecting entry for 198.133.219.0
IP: cache invalidation from 0x115248 0x1378A, new version 5738
RT: add 198.133.219.0 255.255.255.0 via 198.92.76.30, igrp metric [100/10816]
RT: delete route to 198.133.219.0 via 198.92.76.30, igrp metric [100/10816]
RT: no routes to 198.133.219.0, entering holddown
IP: cache invalidation from 0x115248 0x1378A, new version 5739
RT: 198.133.219.0 came out of holddown
RT: garbage collecting entry for 198.133.219.0
IP: cache invalidation from 0x115248 0x1378A, new version 5740
RT: add 198.133.219.0 255.255.255.0 via 198.92.76.30, igrp metric [100/16200]
RT: metric change to 198.133.219.0 via 198.92.76.30, igrp metric [100/16200]
        new metric [100/10816]
RT: delete route to 198.133.219.0 via 198.92.76.30, igrp metric [100/10816]
RT: no routes to 198.133.219.0, entering holddown
IP: cache invalidation from 0x115248 0x1378A, new version 5741
```

S2878

Figure 2-38 Sample Debug IP Routing Output

Explanations for representative lines of output in Figure 2-38 follow.

In the following lines, a newly created entry has been added to the IP routing table. The *metric change* indicates this entry existed previously, but its metric changed and that the metric 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 number that vary from version to version and software load to software load.

```
RT: add 198.93.168.0 255.255.255.0 via 198.92.76.30, igrp metric [100/3020]
RT: metric change to 198.93.168.0 via 198.92.76.30, igrp metric [100/3020]
        new metric [100/2930]
IP: cache invalidation from 0x115248 0x1378A, new version 5736
```

In the following output, the *holddown* and *invalid* 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 *invalid*. *Invalid* corresponds to *came out of holddown*. *Delete route* is triggered when a better path comes along. It gets rid of the old worse path.

```
RT: delete route to 198.133.219.0 via 198.92.76.30, igrp metric [100/10816]
RT: no routes to 198.133.219.0, entering holddown
IP: cache invalidation from 0x115248 0x1378A, new version 5737
RT: 198.133.219.0 came out of holddown
```

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.

debug ip security
no debug ip security

Syntax Description

This command has no arguments or keywords.

Command Mode

EXEC

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 whether or not 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 users on the system will not be adversely affected.

Sample Display

Figure 2-39 shows sample **debug ip security** output.

```
router# debug ip security
IP Security: src 198.92.72.52 dst 198.92.72.53, number of BSO 1
     idb: NULL
     pak: insert (0xFF) 0x0
IP Security: BSO postroute: SECINSERT changed to secret (0x5A) 0x10
IP Security: src 198.92.72.53 dst 198.92.72.52, 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
```

Figure 2-39 Sample Debug IP Security Output

Table 2-22 describes significant fields shown in Figure 2-39.

Table 2-22 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 2-39 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 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.

debug ip tcp driver
no debug ip tcp driver

Syntax Description

This command has no arguments or keywords.

Command Mode

EXEC

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, 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 2-40 shows sample **debug ip tcp driver** output.

```
router# debug ip tcp driver

TCPDRV359CD8: Active open 160.89.80.26:0 --> 160.89.80.25:1996 OK, lport 36628
TCPDRV359CD8: enable tcp timeouts
TCPDRV359CD8: 160.89.80.26:36628 --> 160.89.80.25:1996 Abort
TCPDRV359CD8: 160.89.80.26:36628 --> 160.89.80.25:1996 DoClose tcp abort
```

S2672

Figure 2-40 Sample Debug IP TCP Driver Output

Explanations for individual lines of output from Figure 2-40 follow.

Table 2-23 describes the fields in the first line of output.

Table 2-23 Debug IP TCP Driver Field Descriptions

Field	Description
TCPDRV359CD8:	Unique identifier for this instance of TCP driver activity.
Active open 160.89.80.26	Indicates that the router at IP address 160.89.80.26 has initiated a connection to another router.
:0	TCP port number the initiator of the connection uses to indicate that any port number can be used to set up a connection.
--> 160.89.80.25	Indicates the IP address of the remote router to which the connection has been initiated.
:1996	Indicates 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,	Indicates 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	Indicates that 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 160.89.80.26 (and using TCP port number 36628) is requesting that the connection to IP address 160.89.80.25 using TCP port number 1996 be aborted:

```
TCPDRV359CD8: 160.89.80.26:36628 --> 160.89.80.25:1996 Abort
```

The following line indicates that this connection was in fact closed due to an abort:

```
TCPDRV359CD8: 160.89.80.26:36628 --> 160.89.80.25:1996 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 TCP driver performs. The **no** form of this command disables debugging output.

debug ip tcp driver-pak
no debug ip tcp driver-pak

Syntax Description

This command has no arguments or keywords.

Command Mode

EXEC

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, and X.25 switching currently use the TCP driver.

To observe the context within which certain **debug ip tcp driver-pak** messages occur, turn this command on 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. Using this command not only places a significant load on the system processor, but it may even change the behavior of any bugs that could occur.

Sample Display

Figure 2-41 shows 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)
```

S2673

Figure 2-41 Sample Debug IP TCP Driver-Pak Output

Explanations for individual lines of output from Figure 2-40 follow.

Table 2-24 describes the fields shown in the first line of output.

Table 2-24 Debug TCP Driver-Pak Field Descriptions

Field	Description
TCPDRV359CD8	Unique identifier for this instance of TCP driver activity.
send	Indicates 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	Indicates 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 TCP transactions such as state changes, retransmissions, and duplicate packets. The **no** form of this command disables debugging output.

debug ip tcp transactions
no debug ip tcp transactions

Syntax Description

This command has no arguments or keywords.

Command Mode

EXEC

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** command displays output for packets the router sends and receives, but does not display output for packets it forwards.

Sample Display

Figure 2-41 shows sample **debug ip tcp transactions** output.

```
router# debug ip tcp transactions

TCP: sending SYN, seq 168108, ack 88655553
TCP0: Connection to 26.9.0.13:22530, advertising MSS 966
TCP0: state was LISTEN -> SYNRCVD [23 -> 26.9.0.13(22530)]
TCP0: state was SYNSENT -> SYNRCVD [23 -> 26.9.0.13(22530)]
TCP0: Connection to 26.9.0.13:22530, received MSS 956
TCP0: restart retransmission in 5996
TCP0: state was SYNRCVD -> ESTAB [23 -> 26.9.0.13(22530)]
TCP2: restart retransmission in 10689
TCP2: restart retransmission in 10641
TCP2: restart retransmission in 10633
TCP2: restart retransmission in 13384 -> 26.0.0.13(16151)]
TCP0: restart retransmission in 5996 [23 -> 26.0.0.13(16151)]
```

Figure 2-42 Sample Debug IP TCP Output

Table 2-25 describes significant fields shown in Figure 2-41.

Table 2-25 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 26.9.0.13:22530	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.
state was LISTEN -> SYNSENT	Indicates that the TCP state machine changed state from LISTEN to SYNSENT. Possible TCP states follow: CLOSED—Connection closed. CLOSEWAIT—Received a FIN segment. CLOSING—Received a FIN/ACK segment. ESTAB—Connection established. FINWAIT 1—Sent a FIN segment to start closing the connection. FINWAIT 2—Waiting for a FIN segment. LASTACK—Sent a FIN segment in response to a received FIN segment. LISTEN—Listening for a connection request. SYNRCVD—Received a SYN psegment, and responded. SYNSENT—Sent a SYN segment to start connection negotiation. 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.
[23 -> 26.9.0.13(22530)]	Within these brackets: The first field (23) indicates local TCP port. The second field (26.9.0.13) indicates the destination IP address. The third field (22530) indicates the destination TCP port.
restart retransmission in 5996	Indicates the number of milliseconds until the next retransmission takes place.