CHAPTER 2

Debug Command Listing

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 of the various **debug** commands vary. 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 it to a terminal. For more information about redirecting output, see the chapter, "Using Debug Commands."

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*]	*interface* and *unit* are optional arguments to specify that information for a particular interface be displayed. For example, Ethernet0 specifies the first Ethernet interface; Ethernet1 specifies the second Ethernet interface. If you include this parameter, you must specifiy 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 1-1 shows sample **debug apple arp** output.

Figure 1-1 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

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

The following line of output 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 of output 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 of output 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	*interface* and *unit* are optional arguments to specify that information for a particular interface be displayed. For example, Ethernet0 specifies the first Ethernet interface; Ethernet1 specifies the second Ethernet interface. If you include this parameter, you must specifiy both the interface type and unit number.

Command Mode

EXEC

Usage Guidelines

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

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

Sample Display

Figure 1-2 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.

Figure 1-2 Sample Debug Apple Errors Output

```
router# debug apple errors

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

S2651

As Figure 1-2 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 event**s, the following message can be generated:

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

In the previous 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 previous 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 previous 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 parameter, you must specifiy 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 also will 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 1-3 shows sample **debug apple events** output that describes a nonseed router coming up in discovery mode.

Figure 1-3 Sample Debug Apple Events Output with Discovery Mode State Changes

As Figure 1-3 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 1-3 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:

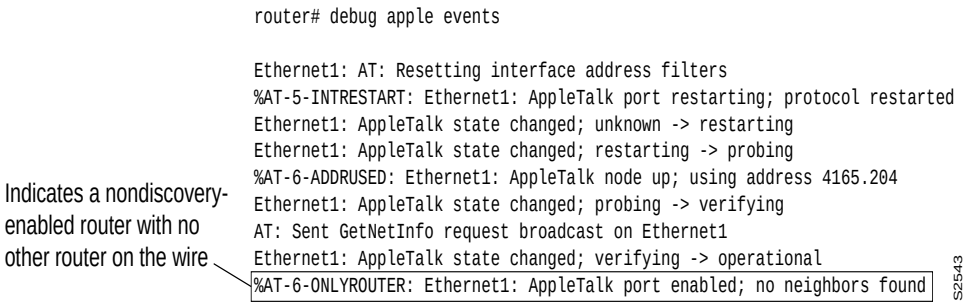
```
Ether0: AppleTalk state changed; verifying -> checking zones
```

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 1-4 shows sample **debug apple events** output that describes a nondiscovery-enabled router coming up when no other router is on the wire.

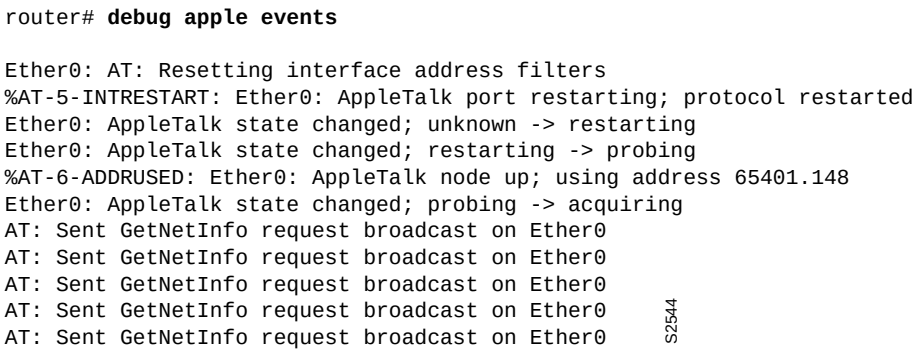
Figure 1-4 Sample Debug Apple Events Output Showing Seed Coming Up by Itself



As Figure 1-4 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 1-4 indicates this situation.

Figure 1-5 shows sample **debug apple events** output that describes a discovery-enabled router coming up when there is no seed router on the wire.

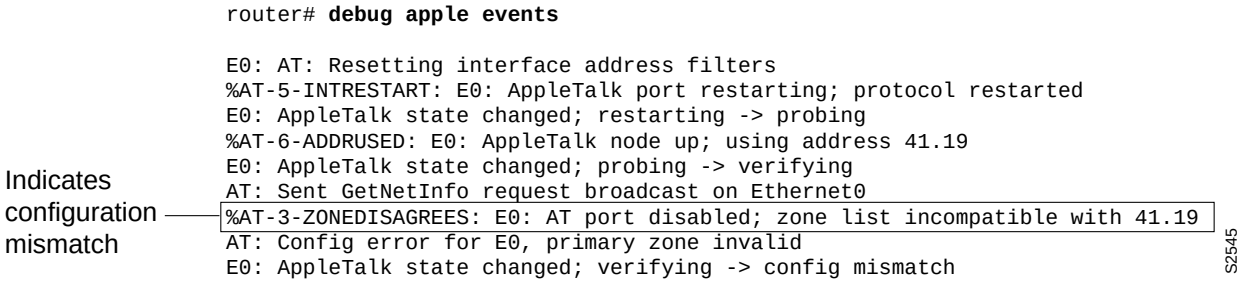
Figure 1-5 Sample Debug Apple Events Output Showing Nonseed with No Seed



As Figure 1-5 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 1-6 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.

Figure 1-6 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 1-6.

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

The router shown in Figure 1-6 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 parameter, you must specifiy 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 1-7 shows sample **debug apple nbp** output.

Figure 1-7 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
```

S2652

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

Table 1-1 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 1-2 describes the fields in the second line of output shown in Figure 1-7.

Table 1-2 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 1-7 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 1-7 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 parameter, you must specifiy 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 1-8 shows sample **debug apple packet** output.

Figure 1-8 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
```

S2653

Table 1-3 describes the fields in the first line of output shown in Figure 1-8.

Table 1-3 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 1-4 describes the fields in the second line of output shown in Figure 1-8.

Table 1-4 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 1-8 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*]	*interface* and *unit* are optional arguments to specify that information for a particular interface be displayed. For example, Ethernet0 specifies the first Ethernet interface; Ethernet1 specifies the second Ethernet interface. If you include this parameter, you must specifiy 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 1-9 shows sample **debug apple routing** output.

Figure 1-9 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

Explanations for representative lines of the **debug apple routing** output in Figure 1-9 follow.

Table 1-5 describes the fields in the first line of sample **debug apple routing** output.

Table 1-5 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 1-5 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 1-6 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	Indicates the number of routes in this RTMP update packet that the router did not already know about.
old 94	Indicates the 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 parameter, you must specifiy 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 1-10 shows sample **debug apple zip** output.

Figure 1-10 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

Explanations of the lines of output shown in Figure 1-10 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) protocol 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 1-11 shows sample **debug arp** output.

Figure 1-11 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
```

S2656

In Figure 1-11, 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 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 1-12 shows sample **debug broadcast** output. Notice how similar it is to the **debug packet** output.

Figure 1-12 Sample Debug Broadcast 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

Table 1-7 describes significant fields shown in Figure 1-12.

Table 1-7 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 1-13 shows sample **debug clns esis events** output.

Figure 1-13 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
```

S2658

Explanations for individual lines of output from Figure 1-13 follow.

The following line of output 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 for this entry is 30.

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

The following line of output 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 (or number of seconds to consider this entry valid before deleting it) is 150.

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

The following line of output 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 1-14 shows sample **debug clns esis packets** output.

Figure 1-14 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
```

S2659

Explanations for individual lines of output from Figure 1-14 follow.

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

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

The following line of output 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 information 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 of output 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 information 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 occuring 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 1-15 shows sample **debug clns events** output.

Figure 1-15 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

Explanations for individual lines of output from Figure 1-15 follow.

The following line of output 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 of output 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 of output 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 of output 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 1-16 shows sample **debug clns igrp packets** output.

Figure 1-16 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                                          S2661
ISO-IGRP: Received level 1 adv for 3333.3333.3333 metric 1100
```

Explanations for individual lines of output from Figure 1-16 follow.

The following line of output 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 of output 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 of output 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 of output 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 of output 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 of output 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 1-17 shows sample **debug clns packet** output.

Figure 1-17 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

Explanations for individual lines of output from Figure 1-17 follow.

In the following lines of output, 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 of output, 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 1-18 shows sample **debug clns routing** output.

Figure 1-18 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
```

S2663

Explanations for individual lines of output from Figure 1-18 follow.

The following line of output 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 of output 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 of output 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 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 then can 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 1-19 shows sample **debug decnet connects** output.

Figure 1-19 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

Table 1-8 describes significant fields shown in Figure 1-19.

Table 1-8 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 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 1-20 shows sample **debug decnet packet** output.

Figure 1-20 Sample Debug DECnet Packet Output

```
router# debug decnet packet
DNET-PKT: src 1.3 dst 1.10 sending to PHASEV
DNET-PKT: Packet forwarded from 1.3 to 1.23
```

S2665

Explanations for individual lines of output from Figure 1-20 follow.

The following line of output 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 of output indicates that the router forwarded a packet from node 1.3 to node 1.23.

```
DNET-PKT: Packet forwarde from 1.3 to 1.23
```

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 1-21 shows sample **debug decnet routing** output.

Figure 1-21 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
```

S2666

Explanations for individual lines of output from Figure 1-21 follow.

The following line of output indicates that the router is sending level 1 updates on interface Ethernet 0:

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

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

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

The following line of output 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 of output 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 of output 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 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 1-22 shows sample **debug frame-relay** output.

Figure 1-22 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
```

S2667

Table 1-9 describes significant fields shown in Figure 1-22.

Table 1-9 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 10MB net 0x0806—IP ARP 0x0808—Frame Relay ARP 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 10MB 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 1-23 shows sample **debug frame-relay events** output.

Figure 1-23 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

As Figure 1-23 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

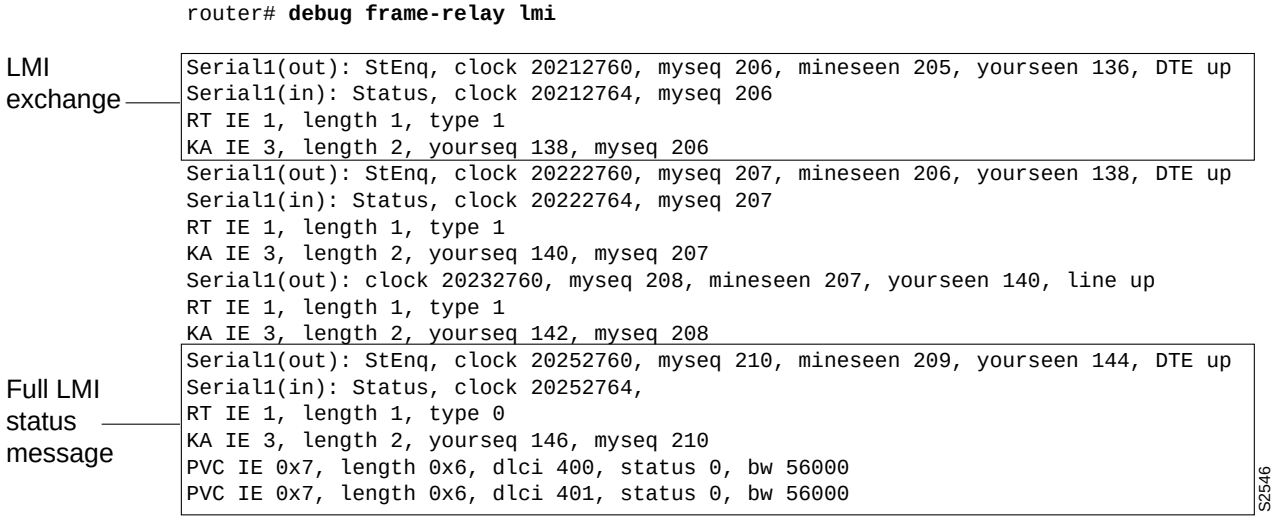
You can use this command to determine whether the router and the Frame Relay switch are sending and receiving LMI packets properly.

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 1-24 shows sample **debug frame-relay lmi** output.

Figure 1-24 Sample Debug Frame-Relay LMI Output



In Figure 1-24, 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 1-24 comprise a full LMI status message that includes a description of the router's two Permanent Virtual Circuits (PVCs).

Table 1-10 describes significant fields in the first line of the **debug frame-relay lmi** output shown in Figure 1-24.

Table 1-10 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, as described in the Frame Relay Specification with Extensions.
yourseen 136	The yourseen counter maps to the LAST RCVD SEQ counter of the switch, as described in the Frame Relay Specification with Extensions.
DTE up	Indicates the line protocol up/down state for the DTE (user) port.

Table 1-11 describes significant fields in the second and third lines of **debug frame-relay lmi** output shown in Figure 1-24.

Table 1-11 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, as described in the Frame Relay Specification with Extensions.
myseq 206	The myseq counter maps to the router's CURRENT SEQ counter, as described in the Frame Relay Specification with Extensions.

Table 1-12 describes significant fields in the last line of **debug frame-relay lmi** output shown in Figure 1-24.

Table 1-12 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

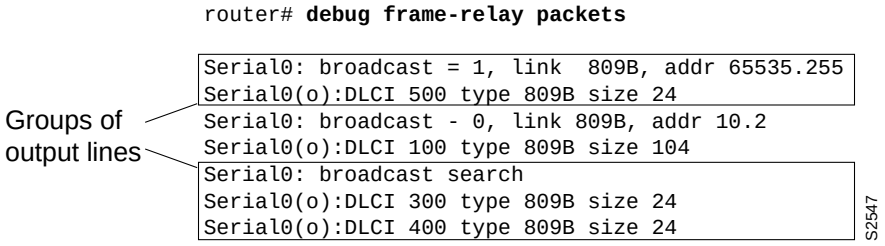
This command helps you to analyze the packets that have been sent on a Frame Relay interface. Because the **debug frame-relay packets** command generates large amount of output, only use it when traffic on the Frame Relay network is less than 25 packets per second.

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

Sample Display

Figure 1-25 shows sample **debug frame-relay packets** output.

Figure 1-25 Sample Debug Frame-Relay Packets Output



As Figure 1-25 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 1-25 describe a single Frame Relay packet that was sent out on two DLCIs.

Table 1-13 describes significant fields shown in the first pair of output lines in Figure 1-25.

Table 1-13 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).

The discussion that follows describes the other lines of **debug frame-relay packet** output shown in Figure 1-25.

The following lines of output 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 of output describe a Frame Relay packet sent to a true broadcast address:

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

The following lines of output 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
```

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 1-26 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 1-26 Sample Debug IP ICMP Output

Table 1-14 describes significant fields shown in the first line of **debug ip icmp** output shown in Figure 1-26.

Table 1-14 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 1-15 describes significant fields shown in the second line of **debug ip icmp** output in Figure 1-26.

Table 1-15 Debug IP ICMP Field Descriptions—Part 2

Field	Description
ICMP:	Indicates that this messages 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 1-14 describes), any of the unreachable messages can have any of the following 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 1-27 shows sample **debug ip igrp events** output.

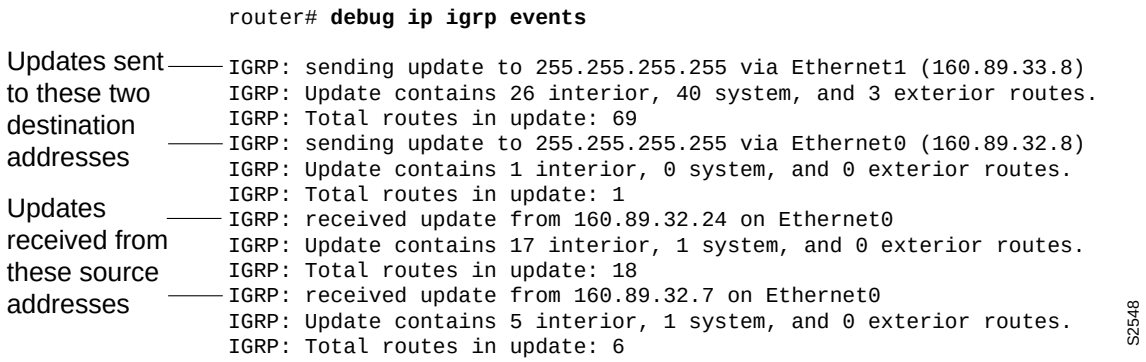


Figure 1-27 Sample Debug IP IGRP Events Output

Figure 1-27 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 1-27 follow.

The first line of output 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 of output 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 of output 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*]	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 1-28 shows sample **debug ip igrp transaction** output.

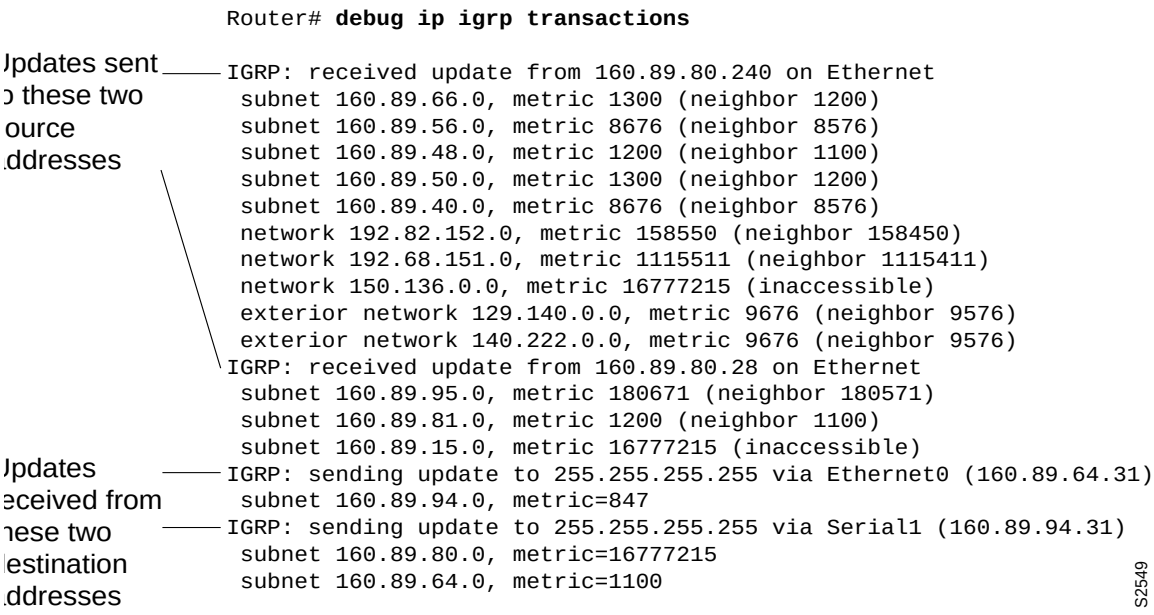


Figure 1-28 Sample Debug IP IGRP Transaction Output

Figure 1-28 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 1-28 is self explanatory.

On the second line in Figure 1-28, 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 1-28 showing 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 1-29 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
```

32670

Figure 1-29 Sample Debug IP OSPF Events Output

The **debug ip ospf events** output shown in Figure 1-29 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 the 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 [*list*]
no debug ip packet [*list*]

Syntax Description

[*list*]	Optional IP access *list* 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 also is sent to the sending host when the router configuration allows it.

Note Because the **debug ip packet** command generates a significant amount 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 1-30 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 1-30 Sample Debug IP Packet Output

Figure 1-30 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 1-16 describes the fields shown in the first line of Figure 1-30.

Table 1-16 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 1-30.

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 1-17 describes.

Table 1-17 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.

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 1-31 shows sample **debug ip rip** output.

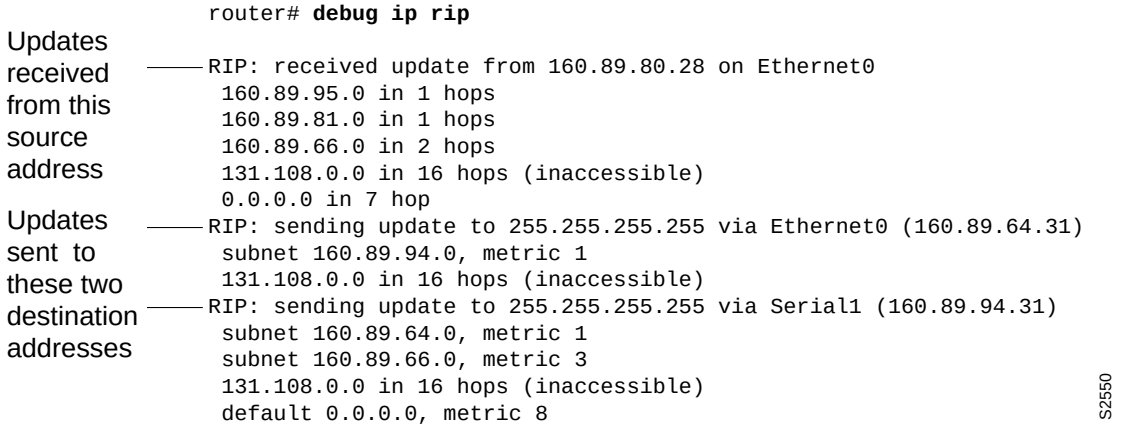


Figure 1-31 Sample Debug IP RIP Output

Figure 1-31 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 1-31 is self-explanatory.

The second line in Figure 1-31 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 1-31 showing that the router is sending updates 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 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 1-32 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 1-32 Sample Debug IP TCP Driver Output

Explanations for individual lines of output from Figure 1-32 follow.

Table 1-18 describes the fields in the following line of output.

```
TCPDRV359CD8: Active open 160.89.80.26:0 --> 160.89.80.25:1996 OK, lport 36628
```

Table 1-18 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 of output 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 of output 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 of output 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 1-33 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 1-33 Sample Debug IP TCP Driver-Pak Output

Explanations for individual lines of output from Figure 1-33 follow.

Table 1-19 describes the fields shown in the following line of output:

```
TCPDRV359CD8: send 2E8CD8 (len 26) queued
```

Table 1-19 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 of output 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 of output 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 1-34 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)]
```

S2674

Figure 1-34 Sample Debug IP TCP Output

Table 1-20 describes significant fields shown in Figure 1-34.

Table 1-20 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.

Field	Description
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 segmnet in response to a received FIN segment. LISTEN—Listening for a connection request. SYNRCVD—Received a SYN psegmnet, 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.

debug ipx packet

Use the **debug ipx packet** EXEC command to display information about packets received, transmitted, and forwarded. The **no** form of this command disables debugging output.

debug ipx packet
no debug ipx packet

Syntax Description

This command has no arguments or keywords.

Command Mode

EXEC

Usage Guidelines

This command is useful for learning whether IPX packets are traveling over a router.

Note In order to generate **debug ipx packet** information on all IPX traffic traveling over the router, you must first configure the router so that fast switching is disabled. Use the **no ipx route-cache** command on all interfaces on which you want to observe traffic. If the router is configured for IPX fast switching, only IPX broadcast packets (SAP, RIP, and Novell NetBIOS) will be displayed.

Sample Display

Figure 1-35 shows sample **debug ipx packet** output.

```
router# debug ipx packet

Novell: src=160.0260.8c4c.4f22, dst=1.0000.0000.0001, packet received
Novell: src=160.0260.8c4c.4f22, dst=1.0000.0000.0001,gw=183.0000.0c01.5d85,
sending packet
```

S2675

Figure 1-35 Sample Debug IPX Packet Output

In Figure 1-35, the first line indicates that the router receives a packet from an Novell station (address 160.0260.8c4c.4f22); this trace does not indicate the address of the immediate router sending the packet to this router. In the second line, the router forwards the packet toward the Novell server (address 1.0000.0000.0001) through an immediate router (183.0000.0c01.5d85).

Table 1-21 describes significant fields shown in Figure 1-35.

Table 1-21 Debug IPX Packet Field Descriptions

Field	Description
IPX	Shows that this is a IPX packet.
src = 160.0260.8c4c.4f22	Source address of the IPX packet. The Novell network number is 160. Its MAC address is 0260.8c4c.4f22.
dst = 1.0000.0000.0001	Destination address for the IPX packet. The address 0000.0000.0001 is an internal MAC address, and the network number 1 is the internal network number of a Novell 3.11 server.
packet received	The router received this packet from a Novell station, possibly through an intermediate router.
gw = 183.0000.0c01.5d85	The router is sending the packet over to the next hop router; its address of 183.0000.0c01.5d85 was learned from the IPX routing table.
sending packet	The router is attempting to send this packet.

debug ipx routing

Use the **debug ipx routing** EXEC command todisplay information on IPX routing packets that the router sends and receives. The **no** form of this command disables debugging output.

debug ipx routing
no debug ipx routing

Syntax Description

This command has no arguments or keywords.

Command Mode

EXEC

Usage Guidelines

Normally, a router or server sends out one routing update per minute. Each routing update packet can include up to 50 entries. If many networks exist on the internetwork, the router sends out multiple packets per update. For example, if a router has 120 entries in the routing table, it would send three routing update packets per update. The first routing update packet would include the first 50 entries, the second packet would include the next 50 entries, and the last routing update packet would include the last 20 entries.

Sample Display

Figure 1-36 shows sample **debug ipx routing** output.

```
router# debug ipx routing

NovellRIP: update from 9999.0260.8c6a.1733
           110801 in 1 hops, delay 2
NovellRIP: sending update to 12FF02:ffff.ffff.ffff via Ethernet 1
           network 555, metric 2, delay 3
           network 1234, metric 3, delay 4
```

S2676

Figure 1-36 Sample Debug IPX Routing Output

Table 1-22 describes significant fields shown in Figure 1-36.

Table 1-22 Debug IPX Routing Field Descriptions

Field	Description
IPXRIP	Shows that this is a IPX RIP packet.
update from 9999.0260.8c6a.1733	Indicates that this packet is a routing update from a Novell server at address 9999.0260.8c6a.1733.
110801 in 1 hops	Indicates that network 110801 is one hop away from the router at address 9999.0260.8c6a.1733.
delay 2	A time measurement (1/18th second) that the NetWare shell uses to estimate how long to wait for a response from a file server. Also known as ticks.

Field	Description
sending update to 12FF02:ffff.ffff.ffff via Ethernet 1	The router is sending this IPX routing update packet to address 12FF02:ffff.ffff.ffff through its Ethernet 1 interface.
network 555	Indicates that the packet includes routing update information for network 555.
metric 2	Indicates that network 555 is two metrics (or hops) away from the router.
delay 3	Indicates that network 555 is a delay of 3 away from the router. Delay is a measurement that the NetWare shell uses to estimate how long to wait for a response from a file server. Also known as ticks.

debug ipx sap

Use the **debug ipx sap** EXEC command to display information about IPX Service Advertisement Protocol (SAP) packets. The **no** form of this command disables debugging output.

debug ipx sap
no debug ipx sap

Syntax Description

This command has no arguments or keywords.

Command Mode

EXEC

Usage Guidelines

Normally, a router or server sends out one SAP update per minute. Each SAP packet can include up to seven entries. If many servers are advertising on the network, the router sends out multiple packets per update. For example, if a router has 20 entries in the SAP table, it would send three SAP packets per update. The first SAP would include the first seven entries, the second SAP would include the next seven entries, and the last update would include the last six entries.

Sample Display

Figure 1-37 shows sample **debug ipx sap** output.

```
router# debug ipx sap

NovellSAP: at 0023F778:
I SAP Response type 0x2 len 160 src:160.0000.0c00.070d dest:160.ffff.ffff.ffff(452)
 type 0x4, “HELLO2”, 199.0002.0004.0006 (451), 2 hops
 type 0x4, “HELLO1”, 199.0002.0004.0008 (451), 2 hops
NovellSAP: sending update to 160
NovellSAP: at 00169080:
 O SAP Update type 0x2 len 96 ssoc:0x452 dest:160.ffff.ffff.ffff(452)
Novell: type 0x4, “Magnolia”, 42.0000.0000.0001 (451), 2 hops
```

Describes a single SAP packet

S2551

Figure 1-37 Sample Debug IPX SAP Output

As Figure 1-37 shows, the **debug ipx sap** command generates multiple lines of output for each SAP packet—a packet summary message and a service detail message.

Explanations for representative lines of output from Figure 1-37 follow.

The first line of output displays the internal router memory address of the packet. The technical support staff uses this information in problem debugging.

```
NovellSAP: at 0023F778:
```

Table 1-23 describes the fields shown in the second line of output in Figure 1-37.

Table 1-23 Debug IPX SAP Field Descriptions—Part 1

Field	Description
I	Indicates whether the router received the SAP packet as input (I) or is sending an update as output (O).
SAP Response type 0x2	Indicates the packet type. Format is 0x*n*; possible values for *n* include: 1—General query 2—General response 3—Get nearest server request 4—Get nearest server response
len 160	Length of this packet (in bytes).
src: 160.000.0c00.070d	Indicates the source address of the packet.
dest:160.ffff.ffff.ffff	Indicates the IPX network number and broadcast address of the destination IPX network for which the message is intended.
(452)	IPX socket number of the process sending the packet at the source address. This number is always 452, which is the socket number for the SAP process.

Table 1-24 describes the fields shown in the thirdand fourth lines of output in Figure 1-37.

Table 1-24 Debug IPX SAP Field Descriptions—Part 2

Field	Description
type 0x4	Indicates the type of service the server sending the packet provides. Format is 0x*n*. Some of the values for *n* are proprietary to Novell. Those values for *n* that have been published include: 0—Unknown 1—User 2—User group 3—Print queue 4—File server 5—Job server 6—Gateway 7—Print server 8—Archive queue 9—Archive server A—Job queue B—Administration 24—Remote bridge server 47—Advertising print server Contact Novell for more information.

Field	Description
“HELLO2”	Name of the server being advertised.
199.0002.0004.0006 (451)	Indicates the network number and address (and socket) of the server generating the SAP packet.
2 hops	Number of hops to the server from the router.

The fifth line of output indicates that the router sent a SAP update to network 160:

```
NovellSAP: sending update to 160
```

As Figure 1-37 shows, the format for **debug ipx sap** output describing a SAP update the router sends is similar to that describing a SAP update the router receives, except that the ssoc: field replaces the src: field, as the following line of output indicates:

```
O SAP Update type 0x2 len 96 ssoc:0x452 dest:160.ffff.ffff.ffff(452)
```

Table 1-25\ describes possible values for the ssoc: field.

Table 1-25 Debug IPX SAP Field Descriptions—Part 3

Field	Description
ssoc:0x452	Indicates the IPX socket number of the process sending the packet at the source address. Possible values include: 451—Network Core Protocol 452—Service Advertising Protocol 453—Routing Information Protocol 455—NetBIOS 456—Diagnostics 4000 to 6000—Ephemeral sockets used for interaction with file servers and other network communications

Related Command

debug ipx routing

debug isdn-event

Use the **debug isdn-event** EXEC command to display ISDN events occurring on the user side (on the router) of the ISDN interface. The ISDN events that may display are Q.931 events (call setup and teardown of ISDN network connections). The **no** form of this command disables debugging output.

debug isdn-event
no debug isdn-event

Syntax Description

This command has no arguments or keywords.

Command Mode

EXEC

Usage Guidelines

Although the debug information provided through the **debug isdn-event** command is similar to the information provided in the **debug isdn-q931** command, the information is displayed in a different format. If you want to see the information displayed in the both formats, you can enable both of these commands at the same time. The displays will be intermingled.

Use the **show dialer** command to retrieve information about the status and configuration of the ISDN interface on the router.

Sample Display

Figure 1-38 shows sample **debug isdn-event** output of call setup events for an outgoing call.

```
router# debug isdn-event

ISDN Event: Call to 415555121202
received HOST_PROCEEDING
 Channel ID i = 0x0101
 ------------------
 Channel ID i = 0x89
received HOST_CONNECT
 Channel ID i = 0x0101
ISDN Event: Connected to 415555121202 on B1 at 64 Kb/s
```

S2677

Figure 1-38 Sample Debug ISDN-Event Output—Call Setup Outgoing Call

Figure 1-39 shows sample **debug isdn-event** output of call setup events for an incoming call. The values used for internal puposes are unpacked information elements. The values that follow the ISDN specification are an interpretation of the unpacked information elements. Refer to Appendix B for information about these values.

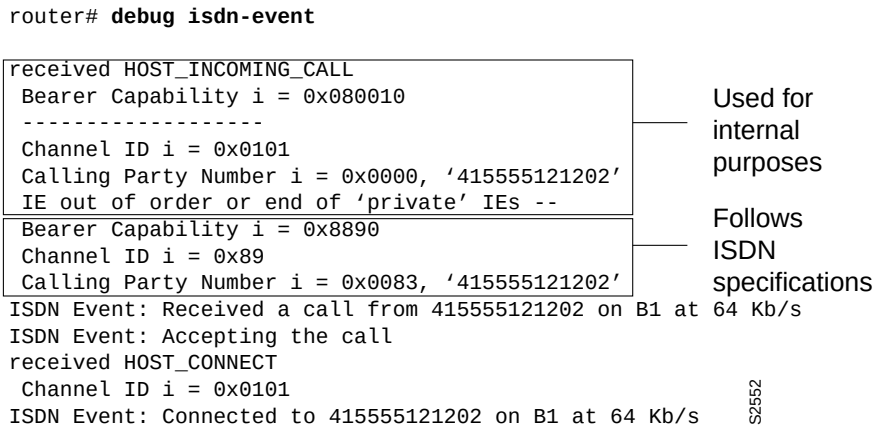


Figure 1-39 Sample Debug ISDN-Event Output—Call Setup Incoming Call

Figure 1-40shows sample **debug isdn-event** output of call teardown events for a call that has been hung up by the other side of the connection.

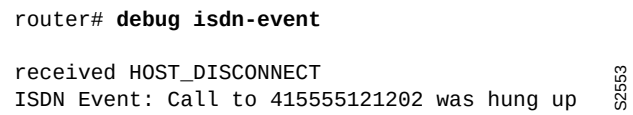


Figure 1-40 Sample Debug ISDN-Event Output—Call Teardown by Destination

Figure 1-41 shows sample **debug isdn- event** output of a call teardown event for an outgoing or incoming call that has been hung up by the ISDN interface on the router side.

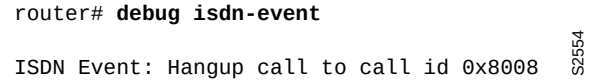


Figure 1-41 Sample Debug ISDN-Event Output—Call Teardown Incoming Call

Table 1-26 describes significant fields shown in Figure 1-38 through Figure 1-41.

Table 1-26 Debug ISDN-Event Field Descriptions

Field	Description
Bearer Capability	Indicates the requested bearer service to be provided by the network.
i=	Indicates the Information Element Identifier. The value depends on the field it is associated with. Refer to the CCITT Q.931 specification for details about the possible values associated with each field for which this identifier is relevant.
Channel ID	Indicates the Channel Identifier. The value 83 indicates any channel, 0101 indicates the B1 channel, and 89 indicates the B1 channel.
Call to	Identifies the called party. This field is only present in outgoing calls. Note that it may be replaced by the Keypad facility field. This field uses the IA5 character set.
IE out of order or end of private' IEs	Indicates that an information element identifier is out of order or there are no more private network information element identifiers to interpret.
Received a call from 415555121202on B1 at 64Kb/s	Identifies the origin of the call. This field is present only in incoming calls. Note that the information about the incoming call includes the channel and speed. Whether this number is displayed depends on the network delivering the calling party number.

debug isdn-q921

Use the **debug isdn-q921** EXEC command to display data link layer (Layer 2) access procedures that are taking place at the router on the D-channel (LAPD) of its Integrated Services Digital Network (ISDN) interface. The **no** form of this command disables debugging output.

debug isdn-q921
no debug isdn-q921

Syntax Description

This command has no arguments or keywords.

Command Mode

EXEC

Usage Guidelines

The ISDN data link layer interface provided by the router conforms to the user interface specification defined by CCITT recommendation Q.921. The display information provided when you enter the **debug isdn-q921** command is limited to commands and responses exchanged during peer-to-peer communication carried over the D-channel. This debug information does not include data transmitted over the B-channels that are also part of the router's ISDN interface. The peers (data link layer entities and layer management entities on the routers) communicate with each other via an ISDN switch over the D-channel.

Note The ISDN switch provides the network interface defined by Q.921. This debug command does not display data link layer access procedures taking place within the ISDN network (that is, procedures taking place on the network side of the ISDN connection). See Appendix B, "ISDN Switch Types, Codes, and Values" for a list of the supported ISDN switch types.

A router can be the calling or called party of the ISDN Q.921 data link layer access procedures. If the router is the calling party, the command displays information about an outgoing call. If the router is the called party, the command displays information about an incoming call and the keepalives (RRs).

The **debug isdn-q921** command can be used with the **debug isdn-event** and the **debug isdn-q931** commands at the same time. The displays will be intermingled. See **debug isdn-event** later in this chapter for samples of combination displays.

Sample Display

Figure 1-42 shows sample **debug isdn-q921** output for an outgoing call.

```
router# debug isdn-q921

471.348 TX -> RRp sapi = 0 tei = 67 nr = 19
471.372 RX <- RRp sapi = 0 tei = 67 nr = 17
471.376 TX -> RRf sapi = 0 tei = 67 nr =19
471.388 RX <- RRf sapi = 0 tei = 67 nr = 17
471.968 TX -> INFOc sapi = 0 tei = 67 ns = 17 nr = 19 i = 0x080105050402889018018
700A80353535313231323032
472.068 RX <- RRr sapi = 0 tei = 67 nr = 18
472.088 RX <- INFOc sapi = 0 tei = 67 ns = 19 nr = 18 i = 0x080185021801 89
472.096 TX -> RRr sapi = 0 tei = 67 nr = 20
472.268 RX <- INFOc sapi = 0 tei = 67 ns = 20 nr 18 i = 0x08018507
472.276 TX -> RRr sapi = 0 tei = 67 nr = 21
472.284 TX -> INFOc sapi = 0 tei = 67 ns 18 nr = 21 i = 0x0801050F
472.356 RX <- RRr sapi = 0 tei = 67 nr = 19
```

Call Setup message

Call Proceeding message

Call Connect message

Connect Ack message

S2555

Figure 1-42 Sample Debug ISDN-Q921 Output for Outgoing Call

Figure 1-43 shows sample **debug isdn-q921** output for an outgoing call.

```
router# debug isdn-q921

139.516 TX -> IDREQ ri = 48386 ai = 127
139.520 RX <- IDREM ri = 0 ai = 89
139.544 RX <- IDASSN ri = 48386 ai = 90
139.552 TX -> SABMEp sapi = 0 tei = 90
139.552 RX <- IDCKRQ ri = 0 ai = 127
139.560 TX -> IDCKRP ri = 36131 ai = 90
140.548 RX <- IDCKRQ ri = 0 ai = 127
140.556 TX -> IDCKRP ri = 24404 ai = 90
140.560 TX -> SABMEp sapi = 0 tei = 90
140.584 RX <- UAf sapi = 0 tei = 90
140.592 TX -> INFOc sapi = 0 tei = 90 ns = 0 nr = 0
 INFORMATION pd = 8 callref = (null)
SPID Information i = 0x34313539303338333636303 1
140.624 RX <- RRr sapi = 0 tei = 90 nr = 1
140.592 RX <- INFOc sapi = 0 tei = 90 ns = 0 nr = 0
 INFORMATION pd = 8 callref = (null)
ENDPOINT IDent i = 0xF080
140.768 TX -> RRr sapi = 0 tei = 90 nr = 1
150.768 TX -> RRp sapi = 0 tei = 90 nr = 1
150.788 RX <- RRf sapi = 0 tei = 90 nr = 1
160.796 TX -> RRp sapi = 0 tei = 90 nr = 1
160.816 RX <- RRf sapi = 0 tei = 90 nr = 1
```

L2 link establishment

S2556

Figure 1-43 Sample Debug ISDN-Q921 Output for Startup Message on a DMS-100 Switch

Figure 1-44 shows sample **debug isdn-q921** output for an incoming call. It is an incoming SETUP message that assumes L2 link is already estatblished to the other side.

```
router# debug isdn-q921

234423.764 TX -> RRp sapi = 0 tei = 66 nr = 36
234423.780 RX <- RRp sapi = 0 tei = 66 nr = 26
234423.784 TX -> RRf sapi = 0 tei = 66 nr = 36
234423.808 RX <- RRf sapi = 0 tei = 66 nr = 26
234425.800 RX <- UAf sapi = 0 tei = 127 i =
0x08010805040288900180018960100083383130313233344536373839303 2
234425.820 TX -> INFOc sapi = 0 tei = 66 ns = 36 nr = 36 i=0x08018807
234425.904 RX <- RRr sapi = 0 tei = 90 nr = 27
234425.920 RX <- INFOc sapi = 0 tei = 66 ns = 36 nr = 33 i=0x0801080F
234433.936 TX -> RRr sapi = 0 tei = 66 nr = 37
234435.940 RX <- RRp sapi = 0 tei = 66 nr = 27
234435.980 TX -> RRf sapi = 0 tei = 66 nr = 37
234435.640 RX <- RRf sapi = 0 tei = 66 nr = 27
```

S2678

Figure 1-44 Debug ISDN-Q921 Output for Incoming Call

Table 1-27 describes significant fields in Figure 1-42, Figure 1-43, and Figure 1-44.

Table 1-27 Debug ISDN-Q921 Field Descriptions

Field	Description
139.516	Indicates the time at which the frame was transmitted from or received by the data link layer entity on the router. The time is maintained by an internal clock. This internal clock is used for the various timers (such as T200, T202, and T201 that may expire while these access procedures are being processed) and for timestamping. Time is in seconds.
TX	Indicates that this frame is being transmitted from the ISDN interface on the local router (user side).
RX	Indicates that this frame is being received by the ISDN interface on the local router from the peer (network side).
IDREQ	Indicates the IdentityRequest message type sent from the local router to the network (assignment source point (ASP)) during the automatic terminal endpoint identifier (TEI) assignment procedure. This message is sent in a UI command frame. The SAPI value for this message type is always 63 (indicating that it is layer 2 management procedure) but it is not displayed. The TEI value for this message type is 127 (indicating that it is a broadcast operation).
ri = 48386	Indicates the Reference number used to differentiate between user devices requesting TEI assignment. This value is a randomly generated number between 0 and 65535. The same ri value sent in the IDREQ message should be returned in the corresponding IDASSN message. Note that a Reference number of 0 indicates that the message is sent from the network side management layer entity and a reference number has not been generated.
ai = 127	Indicates the Action indicator used to request that the ASP assign any TEI value. It is always 127 for the broadcast TEI. Note that in some message types, such as IDREM, a specific TEI value is indicated.

Field	Description
IDREM	Indicates the Identity Remove message type sent from the ASP to the user side layer management entity during the TEI removal procedure. This message is sent in a UI command frame. The ASP sends the Identity Remove message twice to avoid message loss.
IDASSN	Indicates the Identity Assigned message type sent from the ISDN service provider on the network to the local router during the automatic TEI assignment procedure. This message is sent in a UI command frame. The SAPI value for this message type is always 63 (indicating that it is layer 2 management procedure). The TEI value for this message type is 127 (indicating it is a broadcast operation).
ai = 90	Indicates the TEI value automatically assigned by the ASP. This TEI value will be used by data link layer entities on the local router in subsequent communication with the network. The valid values are in the range 64 through 126.
SABME	Indicates the set asynchronous balanced mode extended command. This command places the recipient into modulo 128 multiple frame acknowledged operation. This command also indicates that all exception conditions have been cleared. The SABME command is sent once a second for N200 times (typically three times) until its acceptance is confirmed with a UA response. For a list and brief description of other commands and responses that can be exchanged between the data link layer entities on the local router and the network, see CCITT Recommendation Q.921.
sapi = 0	Identifies the service access point at which the data link layer entity provides services to layer 3 or to the management layer. A SAPI with the value 0 indicates it is a call control procedure. Note that the layer 2 management procedures such as TEI assignment, TEI removal, and TEI checking, which are tracked with the **debug isdn-q921** command, do not display the corresponding SAPI value; it is implicit. If the SAPI value were displayed it would be 63.
tei = 90	Indicates the TEI value automatically assigned by the ASP. This TEI value will be used by data link layer entities on the local router in subsequent communication with the network. The valid values are in the range 64 through 126.
IDCKRQ	Indicates the Identity Check Request message type sent from the ISDN service provider on the network to the local router during the TEI check procedure. This message is sent in a UI command frame. The ri field is always 0. The ai field for this message contains either a specific TEI value for the local router to check or 127, which indicates that the local router should check all TEI values. For a list and brief description of other message types that can be exchanged between the local router and the ISDN service provider on the network, see Appendix B.
IDCKRP	Indicates the Identity Check Response message type sent from the local router to the ISDN service provider on the network during the TEI check procedure. This message is sent in a UI command frame in response to the IDCKRQ message. The ri field is a randomly generated number between 0 and 65535. The ai field for this message contains the specific TEI value that has been checked.
UAf	Confirms that the network side has accepted the SABME command previously sent by the local router. The final bit is set to 1.

Field	Description
INFOc	Indicates that this is an Information command. It is used to transfer sequentially numbered frames containing information fields that are provided by layer 3. The information is transferred across a data link connection.
INFORMATION pd = 8 callref = (null)	Indicates the information fields provided by layer 3. The information is sent one frame at a time. If multiple frames need to be sent, several Information commands are sent. The pd value is the protocol discriminator. The value 8 indicates it is call control information. The call reference number is always null for SPID information,
SPID information i = 0x343135393033383336363031	Indicates the Service Profile IDentifier (SPID). The local router sends this information to the ISDN switch to indicate the services to which it subscribes. SPIDs are assigned by the service provider and are usually 10-digit telephone numbers followed by optional numbers. Currently, only the DMS-100 switch supports SPIDs, one for each B-channel. If SPID information is sent to a switch type other than DMS-100, an error may be displayed in the debug information.
ns = 0	Indicates the send sequence number of transmitted I frames.
nr = 0	Indicates the expected send sequence number of the next received I frame. At time of transmission, this value should be equal to the value of ns. The value of nr is used to determine whether frames need to be retransmitted for recovery.
RRr	Indicates the Receive Ready response for unacknowledged information transfer. The RRr is a response to an INFOc.
RRp	Indicates the Receive Ready command with the poll bit set. The data link layer entity on the user side uses the poll bit in the frame to solicit a response from the peer on the network side.
RRf	Indicates the Receive Ready response with the final bit set. The data link layer entity on the network side uses the final bit in the frame to indicate a response to the poll.
sapi	Indicates the service access point identifier. The SAPI is the point at which data link services are provided to a network layer or management entity. Currently, this field can have the value 0 (for call control procedure) or 63 (for layer 2 management procedures)
tei	Indicates the terminal endpoint identifier (TEI) that has been assigned automatically by the assignment source point (ASP) (also called the layer management entity on the network side). The valid range is 64 through 126. The value 127 indicates a broadcast.

Explanations for individual lines of output from Figure 1-42 follow.

The following lines of output indicate the message exchanges between the data link entity on the local router (user side) and the assignment source point (ASP) on the network side during the TEI assignment procedure. This assumes that the link is down and no TEI currently exists.

```
139.516 TX -> IDREQ ri = 48386 ai = 127
139.544 RX <- IDASSN ri = 48386 ai = 90
```

At 139.516, the local router data link layer entity sent an Identity Request message to the network data link layer entity to request a TEI value that can be used in subsequent communication between the peer data link layer entities. The request includes a randomly generated reference number

(48386) to differentiate between user devices that may be simultaneously requesting automatic TEI assignment and an action indicator of 127 to indicate that the ASP can assign any TEI value available. The ISDN user interface on the router uses automatic TEI assignment.

At 139.544, the network data link entity responds to the Identity Request message with an Identity Assigned message. The response includes the reference number (48386) previously sent in the request and TEI value (90) assigned by the ASP.

The following line of output indicates a message exchange between the layer management entity on the network side and the layer management entity on the local router (user side) during the TEI removal procedure:

```
139.520 RX <- IDREM ri = 0 ai = 89
```

At 139.520, the network layer management entity sends an Identity Remove message when it determines that removal is necessary. The message includes a reference number that is always 0, because it is not responding to a request from the local router. The message also includes the TEI value (89) that is being removed because it is an old value that is no longer used.

The following lines of output indicate the message exchanges between the layer management entity on the network and the layer management entity on the local router (user side) during the TEI check procedure.

```
139.552 RX <- IDCKRQ ri = 0 ai = 127
139.560 TX -> IDCKRP ri = 36131 ai = 90
```

At 139.552, the layer management entity on the network sends the Identity Check Request message to the layer management entity on the local router to check whether a TEI is in use. The message includes a reference number that is always 0 and the TEI value to check. In this case, an ai value of 127 indicates that all TEI values should be checked. At 139.560, the layer management entity on the local router responds with an Identity Check Response message indicating that TEI value 90 is currently in use.

The following lines of output indicate the messages exchanged between the data link layer entity on the local router (user side) and the data link layer on the network side to place the network side into modulo 128 multiple frame acknowledged operation. Note that the data link layer entity on the network side also can initiate the exchange.

```
140.560 TX -> SABMEp sapi = 0 tei = 90
140.584 RX <- UAf sapi = 0 tei = 90
```

At 140.560, the data link layer entity on the local router sends the SABME command with a SAPI of 0 (call control procedure) for TEI 90. At 140.584, the first opportunity, the data link layer entity on the network responds with a UA response. This response indicates acceptance of the command. The data link layer entity sending the SABME command may have to send it more than once before receiving a UA response.

The following lines of output indicate the status of the data link layer entities. Both are ready to receive I frames.

```
150.768 TX -> RRp sapi = 0 tei = 90 nr = 1
150.788 RX <- RRf sapi = 0 tei = 90 nr = 1
```

These I frames are typically exchanged every 10 seconds (T203 timer).

debug isdn-q931

Use the **debug isdn-q931** EXEC command to display information about call setup and teardown of ISDN network connections (layer 3) between the local router (user side) and the network. The **no** form of this command disables debugging output.

debug isdn-q931
no debug isdn-q931

Syntax Description

This command has no arguments or keywords.

Command Mode

EXEC

Usage Guidelines

The ISDN network layer interface provided by the router conforms to the user interface specification defined by CCITT recommendation Q.931 supplemented by other specifications such as for switch types VN2 and VN3.The router tracks only activities that are occurring on the user side, not the network side, of the network connection. The display information provided when you enter the **debug isdn-q931** command is limited to commands and responses exchanged during peer-to-peer communication carried over the D-channel. This debug information does not include data transmitted over the B-channels, which are also part of the router's ISDN interface. The peers (network layers) communicate with each other via an ISDN switch over the D-channel.

A router can be the calling or called party of the ISDN Q.931 network connection call setup and tear-down procedures. If the router is the calling party, the command displays information about an outgoing call. If the router is the called party, the command displays information about an incoming call.

The **debug isdn-q931** command can be used with the **debug isdn-event** and the **debug isdn-q921** commands at the same time. The displays will be intermingled. See **debug isdn-event** earlier in this chapter for samples of combination displays.

Sample Display

Figure 1-45 shows sample **debug isdn-q931** output of a call setup procedure for an outgoing call.

```
router# debug isdn-q931

234191.372 TX -> SETUP pd = 8 callref = 0x04
 Bearer Capability i = 0x8890
 Channel ID i = 0x83
 Called Party Number i = 0x80, '415555121202'
234191.624 RX <- CALL_PROC pd = 8 callref = 0x84
 Channel ID i = 0x89
234191.692 RX <- CONNECT pd = 8 callref = 0x84
234191.692 TX -> CONNECT_ACK pd = 8 callref = 0x04....
Success rate is 0 percent (0/5)
```

S2679

Figure 1-45 Sample Debug ISDN-Q931 Output—Call Setup Procedure for an Outgoing Call

Figure 1-46 shows sample **debug isdn-q931** output of a call setup procedure for an incoming call.

```
router# debug isdn-q931

234223.224 RX <- SETUP pd = 8 callref = 0x06
 Bearer Capability i = 0x8890
 Channel ID i = 0x89
 Calling Party Number i = 0x0083, '81012345678902'
234223.244 TX -> CONNECT pd = 8 callref = 0x86
234223.344 RX <- CONNECT_ACK pd = 8 callref = 0x06
```

S2680

Figure 1-46 Sample Debug ISDN-Q931 Output—Call Setup Procedure for an Incoming Call

Figure 1-47 shows sample **debug isdn-q931** output of a call teardown procedure from the network.

```
router# debug isdn-q931

234207.648 RX <- DISCONNECT pd = 8 callref = 0x84
 Cause i = 0x8790
 Looking Shift to Codeset 6
 Codeset 6 IE 0x1 1 0x82 '10'
234207.668 TX -> RELEASE pd = 8 callref = 0x04
 Cause i = 0x8090
234207.764 RX <- RELEASE_COMP pd = 8 callref = 0x84
```

S2681

Figure 1-47 Sample Debug ISDN-Q931 Output—Call Teardown Procedure from the Network

Figure 1-48 shows sample **debug isdn-q931** output of a call teardown procedure from the router.

```
router# debug isdn-q931

234236.644 TX -> DISCONNECT pd = 8 callref = 0x05
 Cause i = 0x879081
234238.664 RX <- RELEASE pd = 8 callref = 0x85
 Looking Shift to Codeset 6
 Codeset 6 IE 0x1 1 0x82 '10'
234238.752 TX <- RELEASE_COMP pd = 8 callref = 0x05
```

S2682

Figure 1-48 Sample Debug ISDN-Q931 Output—Call Teardown Procedure from the Router

Table 1-28 describes significant fields in Figure 1-45 through Figure 1-48.

Table 1-28 Debug ISDN-Q931 Call Setup Procedure Field Descriptions

Field	Description
234191.372	Indicates the time, in seconds, at which the message was transmitted from or received by the network layer on the router. The time is maintained by an internal clock. This internal clock is used for timeout purposes and timestamping.
TX	Indicates that this message is being transmitted from the local router (user side) to the network side of the ISDN interface.
RX	Indicates that this message is being received by the user side of the ISDN interface from the network side.
SETUP	Indicates that the SETUP message type has been sent to initiate call establishment between peer network layers. This message can be sent from either the local router or the network.
pd	Indicates the protocol discriminator. The protocol discriminator is used to distinguish messages for call control over the user-network ISDN interface from other CCITT-defined messages including other Q.931messages. The protocol discriminator is always 8 for call control messages such as SETUP.
callref	Indicates the call reference number in hexadecimal. The value of this field indicates the number of calls made from either the router (outgoing calls) or the network (incoming calls). Note that the originator of the SETUP message sets the high-order bit of the call reference number to 0. The destination of the connection sets the high-order bit to 1 in subsequent call control messages, such as the CONNECT message. For example, callref = 0x04 in the request becomes callref = 0x84 in the response.
Bearer Capability	Indicates the requested bearer service to be provided by the network.
i=	Indicates the Information Element Identifier. The value depends on the field it is associated with. Refer to the CCITT Q.931 specification for details about the possible values associated with each field for which this identifier is relevant.
Channel ID	Indicates the Channel Identifier. The value 83 indicates any channel, 89 indicates the B1 channel, and 8A indicates the B2 channel. For more information about the Channel Identifier, refer to CCITT Recommendation Q9.31
Called Party Number	Identifies the called party. This field is only present in outgoing SETUP messages. Note that it can be replaced by the Keypad facility field. This field uses the IA5 character set.
Calling Party Number	Identifies the origin of the call. This field is present only in incoming SETUP messages. This field uses the IA5 character set.
CALL_PROC	Indicates the CALL PROCEEDING message; the requested call setup has begun and no more call setup information will be accepted.
CONNECT	Indicates that the called user has accepted the call.
CONNECT_ACK	Indicates that the calling user acknowledges the called user's acceptance of the call.
DISCONNECT	Indicates either that the user side has requested the network to clear an end-to-end connection or that the network has cleared the end-to-end connection.

Field	Description
Cause	Indicates the cause of the disconnect. Refer to CCITT recommendation Q.931 for detailed information about DISCONNECT cause codes and RELEASE cause codes.
Locking Shift to Codeset 6	Indicates that the next information elements will be interpreted according to information element identifiers assigned in codeset 6. Codeset 6 means that the information elements are specific to the local network.
Codeset 6 IE 0x1 i = 0x82, ‘10’	Indicates charging information. This information is specific to the NTT switch type and may not be sent by other switch types.
RELEASE	Indicates that the sending equipment will be releasing the channel and call reference. The recipient of this message should prepare to release the call reference and channel.
RELEASE_COMP	Indicates that the sending equipment has received a RELEASE message and has now released the call reference and channel.

debug isis adj packets

Use the **debug isis adj packets** EXEC command to display information on all adjacency-related activity such as hello packets sent and received and IS-IS adjacencies going up and down. The **no** form of this command disables debugging output.

debug isis adj packets
no debug isis adj packets

Syntax Description

This command has no arguments or keywords.

Command Mode

EXEC

Sample Display

Figure 1-49 shows sample **debug isis adj packets** output.

```
router# debug isis adj packets

ISIS-Adj: Rec L1 IIH from 0000.0c00.40af (Ethernet0), cir type 3, cir id
BBBB.BBBB.BBBB.01
ISIS-Adj: Rec L2 IIH from 0000.0c00.40af (Ethernet0), cir type 3, cir id
BBBB.BBBB.BBBB.01
ISIS-Adj: Rec L1 IIH from 0000.0c00.0c36 (Ethernet1), cir type 3, cir id
CCCC.CCCC.CCCC.03
ISIS-Adj: Area mismatch, level 1 IIH on Ethernet1
ISIS-Adj: Sending L1 IIH on Ethernet1
ISIS-Adj: Sending L2 IIH on Ethernet1                                    S2683
ISIS-Adj: Rec L2 IIH from 0000.0c00.0c36 (Ethernet1), cir type 3, cir id
BBBB.BBBB.BBBB.03
```

Figure 1-49 Sample Debug ISIS Adj Packets Output

Explanations for individual lines of output from Figure 1-49 follow.

The following line of output indicates that the router received an IS-IS hello packet (IIH) on Ethernet0 from the Level 1 router (L1) at MAC address 0000.0c00.40af. The circuit type is the interface type: 1—Level 1 only; 2—Level 2 only; 3—Level 1/2.

The circuit ID is what the neighbor thinks is the designated router for the interface.

```
ISIS-Adj: Rec L1 IIH from 0000.0c00.40af (Ethernet0), cir type 3, cir id BBBB.BBBB.BBBB.01
```

The following line of output indicates that the router (configured as a Level 1 router) received on Ethernet1 an IS-IS hello packet from a Level 1 router in another area, thereby declaring an area mismatch:

```
ISIS-Adj: Area mismatch, level 1 IIH on Ethernet1
```

The following lines of output indicates that the router (configured as a Level 1/Level 2 router) sent on Ethernet1 a Level 1 IS-IS hello packet, and then a Level 2 IS-IS packet:

```
ISIS-Adj: Sending L1 IIH on Ethernet1
ISIS-Adj: Sending L2 IIH on Ethernet1
```

debug isis spf statistics

Use the **debug isis spf statistics** EXEC command to display statistical information about building routes between intermediate systems (ISs). The **no** form of this command disables debugging output.

debug isis spf statistics
no debug isis spf statistics

Syntax Description

This command has no arguments or keywords.

Command Mode

EXEC

Usage Guidelines

The Intermediate System to Intermediate System Intra-Domain Routing Exchange Protocol (IS-IS) provides routing between ISs by flooding the network with link-state information. IS-IS provides routing at two levels, intra-area (level 1) and intra-domain (level 2.) Level 1 routing allows level 1 ISs to communicate with other level 1 ISs in the same area. Level 2 routing allows level 2 ISs to build an interdomain backbone between level 1 areas by traversing only level 2 ISs. Level 1 ISs only need to know the path to the nearest level 2 IS in order to take advantage of the interdomain backbone created by the level 2 ISs.

The IS-IS protocol uses the Shortest Path First (SPF) routing algorithm to build level 1 and level 2 routes.The **debug isis spf statistics** command will provide information for determining the length of time it takes to place a level 1 IS or level 2 IS on the shortest path tree (SPT) using the IS-IS protocol.

Note The SPF algorithm is also called the Dijkstra algorithm, after the creator of the algorithm.

Sample Display

Figure 1-50 shows sample **debug isis spf statistics** output.

```
router# debug isis spf packets

ISIS-Stats: Compute L1 SPT, Timestamp 2780.328 seconds
ISIS-Stats: Complete L1 SPT, Compute time 0.004, 1 nodes on SPT
ISIS-Stats: Compute L2 SPT, Timestamp 2780.3336 seconds
ISIS-Stats: Complete L2 SPT, Compute time 0.056, 12 nodes on SPT
```

S2684

Figure 1-50 Sample Debug ISIS SPf statistics Output

Table 1-29 describes significant fields shown in Figure 1-50.

Table 1-29 Debug ISDN-Event Field Descriptions

Field	Description
Compute L1 SPT	Indicates that level 1 ISs are to be added to a level 1 area.
Timestamp	Indicates the time at which the SPF algorithm was applied. The time indicates the number of seconds that have elapsed since the system has been up and configured.
Complete L1 SPT	Indicates that the algorithm has completed for level 1 routing.
Compute time	Indicates the time it took to place the ISs on the shortest path tree (SPT).
nodes on SPT	Indicates the number of ISs that have been added.
Compute L2 SPT	Indicates that level 2 ISs are to be added to domain.
Complete L2 SPT	Indicates that the algorithm has completed for level 2 routing.

Explanations for individual lines of output from Figure 1-50 follow.

The following lines of output show the statistical information available for Level 1 ISs:

```
ISIS-Stats: Compute L1 SPT, Timestamp 2780.328 seconds
ISIS-Stats: Complete L1 SPT, Compute time 0.004, 1 nodes on SPT
```

The output indicates that the SPF algorithm was applied 2780.328 seconds after the system was up and configured. Given the existing intra-area topology, it took 4 milliseconds to place one level 1 IS on the SPT.

The following lines of output show the statistical information available for Level 2 ISs:

```
ISIS-Stats: Compute L2 SPT, Timestamp 2780.3336 seconds
ISIS-Stats: Complete L2 SPT, Compute time 0.056, 12 nodes on SPT
```

This output indicates that the SPF algorithm was applied 2780.3336 seconds after the system was up and configured. Given the existing intra-domain topology, it took 56 milliseconds to place 12 level 2 ISs on the SPT.

debug isis update-packets

Use the **debug isis update-packets** EXEC command to display various sequence number protocol data units (PDUs) and link state packets that are seen by the router. This router has been configured for IS-IS routing. The **no** form of this command disables debugging output.

debug isis update-packets
no debug isis update-packets

Syntax Description

This command has no arguments or keywords.

Command Mode

EXEC

Sample Display

Figure 1-51 shows sample **debug isis update-packets** output.

```
router# debug isis update-packets

ISIS-Update: Sending L1 CSNP on Ethernet0
ISIS-Update: Sending L2 CSNP on Ethernet0
ISIS-Update: Updating L2 LSP
ISIS-Update: Delete link 888.8800.0181.00 from L2 LSP 1600.8906.4022.00-00, seq E
ISIS-Update: Updating L1 LSP
ISIS-Update: Sending L1 CSNP on Ethernet0
ISIS-Update: Sending L2 CSNP on Ethernet0
ISIS-Update: Add link 8888.8800.0181.00 to L2 LSP 1600.8906.4022.00-00, new seq 10,
 len 91
ISIS-Update: Sending L2 LSP 1600.8906.4022.00-00, seq 10, ht 1198 on Tunnel0
ISIS-Update: Sending L2 CSNP on Tunnel0
ISIS-Update: Updating L2 LSP
ISIS-Update: Rate limiting L2 LSP 1600.8906.4022.00-00, seq 11 (Tunnel0)
ISIS-Update: Updating L1 LSP
ISIS-Update: Rec L2 LSP 888.8800.0181.00.00-00 (Tunnel0)
ISIS-Update: PSNP entry 1600.8906.4022.00-00, seq 10, ht 1196
```

S2685

Figure 1-51 Sample Debug ISIS Update-Packets Output

Explanations for individual lines of output from Figure 1-51 follow.

The following lines of output indicate that the router has sent a periodic level 1 and level 2 complete Sequence Number PDU on Ethernet 0.

```
ISIS-Update: Sending L1 CSNP on Ethernet0
ISIS-Update: Sending L2 CSNP on Ethernet0
```

The following lines of output indicate that the network service access point (NSAP) identified as 8888.8800.0181.00 has been deleted from the level 2 LSP 1600.8906.4022.00-00. The sequence number associated with this LSP is 0xE.

```
ISIS-Update: Updating L2 LSP
ISIS-Update: Delete link 888.8800.0181.00 from L2 LSP 1600.8906.4022.00-00, seq E
```

The following lines of output indicate that the NSAP identified as 8888.8800.0181.00 has been added to the level 2 LSP 1600.8906.4022.00-00. The new sequence number associated with this LSP is 0x10.

```
ISIS-Update: Updating L1 LSP
ISIS-Update: Sending L1 CSNP on Ethernet0
ISIS-Update: Sending L2 CSNP on Ethernet0
ISIS-Update: Add link 8888.8800.0181.00 to L2 LSP 1600.8906.4022.00-00, new seq 10,
 len 91
```

The following line of output indicates that the router has sent level 2 LSP 1600.8906.4022.00-00 with sequence number 0x10 on Tunnel0:

```
ISIS-Update: Sending L2 LSP 1600.8906.4022.00-00, seq 10, ht 1198 on Tunnel0
```

The following lines of output indicates that a level 2 LSP could not be transmitted because it was recently transmitted:

```
ISIS-Update: Sending L2 CSNP on Tunnel0
ISIS-Update: Updating L2 LSP
ISIS-Update: Rate limiting L2 LSP 1600.8906.4022.00-00, seq 11 (Tunnel0)
```

The following lines of output indicate that a level 2 Partial Sequence Number (PSNP) PDU has been received on Tunnel0:

```
ISIS-Update: Updating L1 LSP
ISIS-Update: Rec L2 PSNP from 8888.8800.0181.00 (Tunnel0)
```

The following line of output indicates that a level 2 PSNP PDU with an entry for level 2 LSP 1600.8906.4022.00-00 has been received. This output is an acknowledgment that a previously sent LSP was received without an error.

```
ISIS-Update: PSNP entry 1600.8906.4022.00-00, seq 10, ht 1196
```

debug lapb

Use the **debug lapb** EXEC command to display all traffic for interfaces using LAPB encapsulation. The **no** form of this command disables debugging output.

debug lapb
no debug lapb

Syntax Description

This command has no arguments or keywords.

Command Mode

EXEC

Usage Guidelines

This command displays information on the X.25 layer 2 protocol. It is useful to users who are familiar with the LAPB protocol.

You can use the **debug lapb** command to determine why X.25 virtual circuits or LAPB connections are going up and down. It is also useful for identifying link problems, as evidenced when **show interfaces** command displays a high number of rejects or frame errors over the X.25 link.

Caution Because the **debug lapb** command generates a lot of output, use it when the aggregate of all LAPB traffic on X.25 and LAPB interfaces is fewer than five frames per second.

Sample Display

Figure 1-52 shows sample **debug lapb** output. (The numbers 1 through 6 at the top of the display have been added in order to aid documentation.)

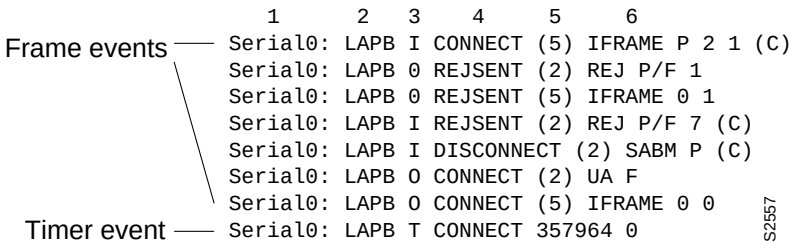


Figure 1-52 Sample Debug LAPB Output

In Figure 1-52 each line of output describes a LAPB event. There are two types of LAPB events: frame events (when a frame enters or exits the router) and timer events. In Figure 1-52, the last line describes a timer event; all of the other lines describe frame events. Table 1-30 describes the first six fields shown in Figure 1-52.

Table 1-30 Debug LAPB Field Descriptions

Field	Description
First field	Interface type and unit number reporting the frame event.
Second field	Protocol providing the information.
Third field	Command Mode of frame event. Possible values follow: I—Frame input O—Frame output T—T1 timer expired
Fourth field	State of the protocol when the frame event occurred. Possible values follow: BUSY (RNR frame received) CONNECT DISCONNECT DISCSENT (disconnect sent) ERROR (FRMR frame sent) REJSENT (reject frame sent) SABMSENT (SABM frame sent)
Fifth field	In a frame event, this value is the size of the frame (in bytes). In a timer event, this value is the current timer value (in milliseconds).
Sixth field	In a frame event, this value is the frame type name. Possible values for frame type names follow: DISC—Disconnect DM—Disconnect mode FRMR—Frame reject IFRAME—Information frame ILLEGAL—Illegal LAPB frame REJ—Reject RNR—Receiver not ready RR—Receiver ready SABM—Set asynchronous balanced mode UA—Unnumbered acknowledgment In a timer event, this value is the number of retransmissions already attempted.

AsFigure 1-52 shows, a timer event only displays the first six fields of **debug lapb** output. For frame events, however, the fields that follow the sixth field document the LAPB control information present in the frame. Depending on the value of the frame type name shown in the sixth field, these fields may or may not appear. Descriptions of the fields following the first six fields shown in Figure 1-52 follow.

If the frame's Poll/Final bit is set, an indicator will be printed after the frame type name. Possible values follow:

- F—Final (printed for Response frames)
- P—Poll (printed for Command frames)
- P/F—Poll/Final (printed for RR, RNR and REJ frames, which can be either Command or Response frames)

After the Poll/Final indicator, depending on the frame type, three different types of LAPB control information can be printed.

For information frames, the value of the N(S) field and the N(R) field will be printed. The N(S) field of an information frame is the sequence number of that frame, so this field will rotate between 0 and 7 for successive outgoing information frames and (under normal circumstances) also will rotate for incoming information frame streams. The N(R) field is a "piggybacked" acknowledgment for the incoming information frame stream; it informs the other end of the link what sequence number is expected next.

RR, RNR, and REJ frames have an N(R) field, so the value of that field is printed. This field has exactly the same significance that it does in an information frame.

For the FRMR frame, the frame's three bytes of error information is printed (in hexadecimal notation).

The remaining frames do not have this data, so nothing will be printed.

For incoming frames, the last field will indicate whether the received frame was a command (C) or a response (R).

debug lat packet

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

debug lat packet
no debug lat packet

Syntax Description

This command has no arguments or keywords.

Command Mode

EXEC

Usage Guidelines

For each datagram (packet) received or transmitted, a message is logged to the console.

Note This command severely impacts LAT performance and is intended for troubleshooting use only.

Sample Display

Figure 1-53 shows sample **debug lat packet** output.

```
router# debug lat packet

LAT: I int=Ethernet0, src=0000.0c01.0509, dst=0900.2b00.000f, type=0, M=0, R=0
LAT: I int=Ethernet0, src=0800.2b11.2d13, dst=0000.0c01.7876, type=A, M=0, R=0
LAT: O dst=0800.2b11.2d13, int=Ethernet0, type= A, M=0, R=0, len= 20, next 0 ref 1
```

S2686

Figure 1-53 Sample Debug LAT Packet Output

The following line of output describes a packet that is input to the router. Table 1-31 describes the fields in this line of output.

```
LAT: I int=Ethernet0, src=0800.2b11.2d13, dst=0000.0c01.7876, type=A, M=0, R=0
```

Table 1-31 Debug LAT Packet Field Descriptions

Field	Description
LAT:	Indicates that this display shows LAT debugging output.
I	Indicates that this line of output describes a packet that is input to the router (I) or output from the router (O).
int = Ethernet0	Indicates the interface on which the packet event took place.
src = 0800.2b11.2d13	Indicates the source address of the packet.
dst = 0000.0c01.7876	Indicates the destination address of the packet.
type = 0	Indicates the message type (in hex). Possible values are: 0 = Run Circuit 1 = Start Circuit 2 = Stop Circuit A = Service Announcement C = Command D = Status E = Solicit Information F = Response Information

The following line of output describes a packet that is output from the router. Table 1-32 describes the last three fields in this line of output.

```
LAT: O dst=0800.2b11.2d13, int=Ethernet0, type= A, M=0, R=0, len= 20, next 0 ref 1
```

Table 1-32 Debug LAT Packet Field Descriptions

Field	Description
len= 20	Indicates the length (hex) of the packet in bytes.
next 0	Indicates the link on transmit queue.
ref 1	Indicates the count of packet users.

debug lnm events

Use the **debug lnm events** EXEC command to display any unusual events that occur on a Token Ring network. These events includes such as stations reporting errors, or error thresholds being exceeded. The **no** form of this command disables debugging output.

debug lnm events
no debug lnm events

Syntax Description

This command has no arguments or keywords.

Command Mode

EXEC

Sample Display

Figure 1-54 shows sample **debug lnm events** output.

```
router# debug lnm events

IBMNM3: Adding 0000.3001.1166 to error list
IBMNM3: Station 0000.3001.1166 going into preweight condition
IBMNM3: Station 0000.3001.1166 going into weight condition
IBMNM3: Removing 0000.3001.1166 from error list
LANMGR0: Beaconing is present on the ring
LANMGR0: Ring is no longer beaconing
IBMNM3: Beaconing, Postmortem Started
IBMNM3: Beaconing, heard from 0000.3000.1234
IBMNM3: Beaconing, Postmortem Next Stage
IBMNM3: Beaconing, Postmortem Finished
```

S2687

Figure 1-54 Sample Debug LNM Events Output

Explanations for the messages shown in Figure 1-54 follow.

The following message indicates that station 0000.3001.1166 reported errors and has been added to the list of stations reporting errors. This station is located on Ring 3.

```
IBMNM3: Adding 0000.3001.1166 to error list
```

The following message indicates that station 0000.3001.1166 has passed the “early warning” threshold for error counts:

```
IBMNM3: Station 0000.3001.1166 going into preweight condition
```

The following message indicates that station 0000.3001.1166 is experiencing a severe number of errors:

```
IBMNM3: Station 0000.3001.1166 going into weight condition
```

The following message indicates that the error counts for station 0000.3001.1166 have all decayed to zero, so this station is being removed from the list of stations that have reported errors:

```
IBMNM3: Removing 0000.3001.1166 from error list
```

The following message indicates that Ring 0 has entered failure mode. This ring number is assigned internally.

```
LANMGR0: Beaconing is present on the ring
```

The following message indicates that Ring 0 is no longer in failure mode. This ring number is assigned internally.

```
LANMGR0: Ring is no longer beaconing
```

The following message indicates that the router is beginning its attempt to determine whether or not any stations left the ring during the automatic recovery process for the last beaconing failure. The router attempts to contact stations that were part of the fault domain to see if they are still operating on the ring.

```
IBMNM3: Beaconing, Postmortem Started
```

The following message indicates that the router is attempting to determine whether or not any stations left the ring during the automatic recovery process for the last beaconing failure. It heard back from station 0000.3000.1234, one of the two stations in the fault domain.

```
IBMNM3: Beaconing, heard from 0000.3000.1234
```

The following message indicates that the router is attempting to determine whether or not any stations left the ring during the automatic recovery process for the last beaconing failure. It is initiating another attempt to contact the two stations in the fault domain.

```
IBMNM3: Beaconing, Postmortem Next Stage
```

The following output indicates that the router has attempted to determine whether or not any stations left the ring during the automatic recovery process for the last beaconing failure. It has successfully heard back from both stations that were part of the fault domain.

```
IBMNM3: Beaconing, Postmortem Finished
```

Explanations for other messages that the **debug lnm events** command can generate follow.

The following message indicates that the router is out of memory:

```
LANMGR: memory request failed, find_or_build_station()
```

The following message indicates that Ring 3 is experiencing a large number of errors that cannot be attributed to any individual station:

```
IBMNM3: Non-isolating error threshold exceeded
```

The following message indicates that a station (or stations) on Ring 3 are receiving frames faster than they can be processed.

```
IBMNM3: Adapters experiencing congestion
```

The following message indicates that the beaconing has lasted for over 1 minute and is considered to be a “permanent” error:

```
IBMNM3: Beaconing, permanent
```

The following message indicates that the beaconing lasted for less than 1 minute. The router is attempting to determine whether either of the stations in the fault domain left the ring.

```
IBMNM: Beaconing, Destination Started
```

In the preceding line of output, the following can replace Started: Next State; Finished; Timed out; and Cannot find station 0000.0301.4876.

debug lnm llc

Use the **debug lnm llc** EXEC command to display all communication between the router/bridge and the LNMs that have connections to it. The **no** form of this command disables debugging output.

debug lnm llc
no debug lnm llc

Syntax Description

This command has no arguments or keywords.

Command Mode

EXEC

Usage Guidelines

One line is displayed for each message sent or received.

Sample Display

Figure 1-55 shows sample **debug lnm llc** output.

```
router# debug lnm llc

IBMNM: Received LRM Set Reporting Point frame from 1000.5ade.0d8a.
IBMNM: found bridge: 001-2-00A, addresses: 0000.3040.a630 4000.3040.a630
IBMNM: Opening connection to 1000.5ade.0d8a on TokenRing0
IBMNM: Sending LRM LAN Manager Accepted to 1000.5ade.0d8a on link 0.
IBMNM: sending LRM New Reporting Link Established to 1000.5a79.dbf8 on link 1.
IBMNM: Determining new controlling LNM
IBMNM: Sending Report LAN Manager Control Shift to 1000.5ade.0d8a on link 0.
IBMNM: Sending Report LAN Manager Control Shift to 1000.5a79.dbf8 on link 1.

IBMNM: Bridge 001-2-00A received Request Bridge Status from 1000.5ade.0d8a.
IBMNM: Sending Report Bridge Status to 1000.5ade.0d8a on link 0.
IBMNM: Bridge 001-2-00A received Request REM Status from 1000.5ade.0d8a.
IBMNM: Sending Report REM Status to 1000.5ade.0d8a on link 0.
IBMNM: Bridge 001-2-00A received Set Bridge Parameters from 1000.5ade.0d8a.
IBMNM: Sending Bridge Parameters Set to 1000.5ade.0d8a on link 0.
IBMNM: sending Bridge Params Changed Notification to 1000.5a79.dbf8 on link 1.
IBMNM: Bridge 001-2-00A received Set REM Parameters from 1000.5ade.0d8a.
IBMNM: Sending REM Parameters Set to 1000.5ade.0d8a on link 0.
IBMNM: sending REM Parameters Changed Notification to 1000.5a79.dbf8 on link 1.
IBMNM: Bridge 001-2-00A received Set REM Parameters from 1000.5ade.0d8a.
IBMNM: Sending REM Parameters Set to 1000.5ade.0d8a on link 0.
IBMNM: sending REM Parameters Changed Notification to 1000.5a79.dbf8 on link 1.
IBMNM: Received LRM Set Reporting Point frame from 1000.5ade.0d8a.
IBMNM: found bridge: 001-1-00A, addresses: 0000.3080.2d79 4000.3080.2d7
```

S2688

Figure 1-55 Sample Debug LNM LLC Output

As Figure 1-55 indicates, **debug lnm llc** output can vary somewhat in format. Table 1-33 describes significant fields shown in the first line of output in Figure 1-55.

Table 1-33 Debug LNM LLC Field Descriptions

Field	Description
IBMNM:	Indicates that this line of output displays LLC-level debugging information.
Received	Indicates that the router received a frame. The other possible value is Sending, to indicate that the router is sending a frame.
LRM	Indicates which function of the LLC-level software is communicating: CRS—Configuration Report Server LBS—LAN Bridge Server LRM—LAN Reporting Manager REM—Ring Error Monitor RPS—Ring Parameter Server RS—Ring Station
Set Reporting Point	Name of the specific frame that the router sent or received. Possible values include the following: Bridge Counter Report Bridge Parameters Changed Notification Bridge Parameters Set CRS Remove Ring Station CRS Report NAUN Change CRS Report Station Information CRS Request Station Information CRS Ring Station Removed LRM LAN Manager Accepted LRM Set Reporting Point New Reporting Link Established REM Forward MAC Frame REM Parameters Changed Notification REM Parameters Set Report Bridge Status Report LAN Manager Control Shift Report REM Status Request Bridge Status Request REM Status Set Bridge Parameters Set REM Parameters
from 1000.5ade.0d8a	If the router has received the frame, this address is the source address of the frame. If the router is sending the frame, this address is the destination address of the frame.

Explanations for other types of messages shown in Figure 1-55 follow.

The following message indicates that the lookup for the bridge with which the LAN Manager was requesting to communicate was successful:

```
IBMNM: found bridge: 001-2-00A, addresses: 0000.3040.a630 4000.3040.a630
```

The following message is self-explanatory:

```
IBMNM: Opening connection to 1000.5ade.0d8a on TokenRing0
```

The following message indicates that a LAN Manager has connected or disconnected from an internal bridge and that the router computes which LAN Manager is allowed to change parameters.

```
IBMNM: Determining new controlling LNM
```

The following line of output indicates which bridge in the router is the destination for the frame:

```
IBMNM: Bridge 001-2-00A received Request Bridge Status from 1000.5ade.0d8a.
```

debug lnm mac

Use the **debug lnm mac** EXEC command to display all management communication between the router/bridge and all stations on the local Token Rings. The **no** form of this command disables debugging output.

debug lnm mac
no debug lnm mac

Syntax Description

This command has no arguments or keywords.

Command Mode

EXEC

Usage Guidelines

One line is displayed for each message sent or received.

Sample Display

Figure 1-56 shows sample **debug lnm mac** output.

```
router# debug lnm mac

LANMGR0: RS received request address from 4000.3040.a670.
LANMGR0: RS sending report address to 4000.3040.a670.
LANMGR0: RS received request state from 4000.3040.a670.
LANMGR0: RS sending report state to 4000.3040.a670.
LANMGR0: RS received request attachments from 4000.3040.a670.
LANMGR0: RS sending report attachments to 4000.3040.a670.
LANMGR2: RS received ring purge from 0000.3040.a630.
LANMGR2: CRS received report NAUN change from 0000.3040.a630.
LANMGR2: RS start watching ring poll.
LANMGR0: CRS received report NAUN change from 0000.3040.a630.
LANMGR0: RS start watching ring poll.
LANMGR2: REM received report soft error from 0000.3040.a630.
LANMGR0: REM received report soft error from 0000.3040.a630.
LANMGR2: RS received ring purge from 0000.3040.a630.
LANMGR2: RS received AMP from 0000.3040.a630.
LANMGR2: RS received SMP from 0000.3080.2d79.
LANMGR2: CRS received report NAUN change from 1000.5ade.0d8a.
LANMGR2: RS start watching ring poll.
LANMGR0: RS received ring purge from 0000.3040.a630.
LANMGR0: RS received AMP from 0000.3040.a630.
LANMGR0: RS received SMP from 0000.3080.2d79.
LANMGR0: CRS received report NAUN change from 1000.5ade.0d8a.
LANMGR0: RS start watching ring poll.
LANMGR2: RS received SMP from 1000.5ade.0d8a.
LANMGR2: RPS received request initialization from 1000.5ade.0d8a.
LANMGR2: RPS sending initialize station to 1000.5ade.0d8a.
```

S2689

Figure 1-56 Sample Debug LNM MAC Output

Table 1-34 describes significant fields shown in the first line of output in Figure 1-56.

Table 1-34 Debug LNM MAC Field Descriptions

Field	Description
LANMGR0:	LANMGR indicates that this line of output displays MAC-level debugging information. 0 indicates the number of the Token Ring interface associated with this line of debugging output.
RS	Indicates which function of the MAC-level software is communicating: CRS—Configuration Report Server REM—Ring Error Monitor RPS—Ring Parameter Server RS—Ring Station
received	Indicates that the router received a frame. The other possible value is sending, to indicate that the router is sending a frame.
request address	Name of the specific frame that the router sent or received. Possible values include the following: AMP initialize station report address report attachments report NAUN change report soft error report state request address request attachments request initialization request state ring purge SMP
from 4000.3040.a670	If the router has received the frame, this address is the source address of the frame. If the router is sending the frame, this address is the destination address of the frame.

As Figure 1-56 indicates, all **debug lnm mac** messages follow the format described in Table 1-34 except the following:

```
LANMGR2: RS start watching ring poll
LANMGR2: RS stop watching ring poll
```

These messages indicate that the router starts and stops receiving AMP and SMP frames. These frames are used to build a current picture of which stations are on the ring.

debug local-ack state

Use the **debug local-ack state** EXEC command to display the new and the old state conditions whenever there is a state change in the Local Acknowledgment state machine. The **no** form of this command disables debugging output.

debug local-ack state
no debug local-ack state

Syntax Description

This command has no arguments or keywords.

Command Mode

EXEC

Sample Display

Figure 1-57 shows sample **debug local-ack state** output.

```
router# debug local-ack state

LACK_STATE: 2370300, hashp 2AE628, old state = disconn, new state = awaiting
LLC2 open to finish
LACK_STATE: 2370304, hashp 2AE628, old state = awaiting LLC2 open to finish,
new state = connected
LACK_STATE: 2373816, hashp 2AE628, old state = connected, new state = disconnected
LACK_STATE: 2489548, hashp 2AE628, old state = disconn, new state = awaiting
LLC2 open to finish
LACK_STATE: 2489548, hashp 2AE628, old state = awaiting LLC2 open to finish,
new state = connected
LACK_STATE: 2490132, hashp 2AE628, old state = connected, new state = awaiting
linkdown response
LACK_STATE: 2490140, hashp 2AE628, old state = awaiting linkdown response,
new state = disconnected
LACK_STATE: 2497640, hashp 2AE628, old state = disconn, new state = awaiting
LLC2 open to finish                                                          S2690
LACK_STATE: 2497644, hashp 2AE628, old state = awaiting LLC2 open to finish,
new state = connected
```

Figure 1-57 Sample Debug Local-Ack State Output

Table 1-35 describes significant fields shown in Figure 1-57.

Table 1-35 Debug Local-Ack State Field Descriptions

Field	Description
LACK_STATE:	Indicates that this packet describes a state change in the Local Acknowledgment state machine.
2370300	System clock.
hashp 2AE628	Internal control block pointer used by technical support staff for debugging purposes.
old state = disconn	Indicates the old state condition in the Local Acknowledgment state machine. Possible values include: Disconn (disconnected) awaiting LLC2 open to finish connected awaiting linkdown response
new state = awaiting LLC2 open to finish	Indicates the new state condition in the Local Acknowledgment state machine. Possible values include: Disconn (disconnected) awaiting LLC2 open to finish connected awaiting linkdown response

debug netbios-name-cache

Use the **debug netbios-name-cache** EXEC command to display name caching activities on a router. The **no** form of this command disables debugging output.

debug netbios-name-cache
no debug netbios-name-cache

Syntax Description

This command has no arguments or keywords.

Command Mode

EXEC

Usage Guidelines

Examine the display to diagnose problems in NetBIOS name caching.

Sample Display

Figure 1-58 illustrates a collection of sample **debug netbios-name-cache** display output listings.

```
router# debug netbios-name-cache

NETBIOS: L checking name ORINDA , vrn=0
NetBIOS name cache table corrupted at offset 13
NetBIOS name cache table corrupted at later offset, at location 13
NETBIOS: U chk name=ORINDA, addr=1000.4444.5555, idb=TR1, vrn=0, type=1
NETBIOS: U upd name=ORINDA,addr=1000.4444.5555,idb=TR1,vrn=0,type=1
NETBIOS: U add name=ORINDA,addr=1000.4444.5555,idb=TR1,vrn=0,type=1
NETBIOS: U no memory to add cache entry. name=ORINDA,addr=1000.4444.5555
NETBIOS: Invalid structure detected in netbios_name_cache_ager
NETBIOS: flushed name=ORINDA, addr=1000.4444.5555
NETBIOS: expired name=ORINDA, addr=1000.4444.5555
NETBIOS: removing entry. name=ORINDA,addr=1000.4444.5555,idb=TR1,vrn=0
NETBIOS: Tossing ADD_NAME/STATUS/NAME/ADD_GROUP frame
NETBIOS: Lookup Failed -- not in cache
NETBIOS: Lookup Worked, but split horizon failed
NETBIOS: Could not find RIF entry                                    S2691
NETBIOS: Cannot duplicate packet in netbios_name_cache_proxy
```

Figure 1-58 Sample Debug NetBIOS-Name-Cache Output

Note The sample display provided in Figure 1-58 is a composite output. Debugging output that you might actually see would not necessarily be presented in this sequence.

Descriptions of selected **debug netbios-name-cache** output fields are provided in Table 1-36.

Table 1-36 Debug NetBIOS-Name-Cache Field Descriptions

Field	Description
NETBIOS	Indicates that this is a NetBIOS name caching debugging output.
L, U	L means lookup; U means update.
vrn=0	Router determined that the packet comes from virtual ring number 0; this packet actually comes from a real Token Ring interface, because virtual ring number 0 is not valid.
addr=1000.4444.5555	MAC address 1000.4444.5555 of machine being looked up in NetBIOS name cache.
idb=TR1	Indicates that name of machine was learned from Token Ring interface number 1; idb translates into interface data block
type=1	The type field indicates the way that the router learned about the specified machine. The possible values for type are as follows: 1 = Learned from traffic 2 = Learned from a remote peer 4, 8 = Statically entered via the router's configuration

The following discussion briefly outlines each line shown in the example provided in Figure 1-58.

With the first line of output, the router declares that it has examined the NetBIOS name cache table for the machine name ORINDA and that the packet that prompted the lookup came from virtual ring 0. In this case, this packet comes from a real interface— virtual ring number 0 is not valid.

```
NETBIOS: L checking name ORINDA, vrn=0
```

The following two entries indicate that there is a invalid NetBIOS entry and that the corrupted memory was detected. The invalid memory will be removed from the table; no action is needed.

```
NetBIOS name cache table corrupted at offset 13
NetBIOS name cache table corrupted at later offset, at location 13
```

The following output indicates that the router has attempted to check the NetBIOS cache table for the name ORINDA with MAC address 1000.4444.5555. This name was obtained from Token Ring interface 1. The type field indicates that the name was learned from traffic.

```
NETBIOS: U chk name=ORINDA, addr=1000.4444.5555, idb=TR1, vrn=0, type=1
```

The following display indicates that the NetBIOS name ORINDA is in the name cache table and has been updated to the current value:

```
NETBIOS: U upd name=ORINDA,addr=1000.4444.5555,idb=TR1,vrn=0,type=1
```

The following display indicates that the NetBIOS name ORINDA is not in the table and must be added to the table:

```
NETBIOS: U add name=ORINDA,addr=1000.4444.5555,idb=TR1,vrn=0,type=1
```

The following display indicates that there was insufficient cache buffer space when the router tried to add this name:

```
NETBIOS: U no memory to add cache entry. name=ORINDA,addr=1000.4444.5555
```

The following display indicates that the NetBIOS ager detects an invalid memory in the cache. The router clears the entry; no action is needed.

```
NETBIOS: Invalid structure detected in netbios_name_cache_ager
```

The following display indicates that the entry for ORINDA has been flushed from the cache table:

```
NETBIOS: flushed name=ORINDA, addr=1000.4444.5555
```

The following display indicates that the entry for ORINDA has timed out and has been flushed from the cache table:

```
NETBIOS: expired name=ORINDA, addr=1000.4444.5555
```

The following display indicates that the router has removed the ORINDA entry from its cache table:

```
NETBIOS: removing entry. name=ORINDA,addr=1000.4444.5555,idb=TR1,vrn=0
```

The following display indicates that the router discarded a NetBIOS packet of type ADD_NAME, STATUS, NAME_QUERY, or ADD_GROUP. These packets are discarded when multiple copies of one of these packet types are detected during a certain period of time.

```
NETBIOS: Tossing ADD_NAME/STATUS/NAME/ADD_GROUP frame
```

The following display indicates that the system was unable to find a NetBIOS name in the cache:

```
NETBIOS: Lookup Failed -- not in cache
```

The following display indicates that the destination NetBIOS name was found in the cache but was determined to be located on the same ring from which the packet came. The router would drop this packet because it should not leave this ring.

```
NETBIOS: Lookup Worked, but split horizon failed
```

The following display indicates that the NetBIOS name was found in the cache but the router could not find the corresponding RIF. The packet will be sent as a broadcast frame.

```
NETBIOS: Could not find RIF entry
```

The following display indicates that no buffer was available to create a NetBIOS name-cache proxy. A proxy will not be created for the packet, which will be forwarded as a broadcast frame.

```
NETBIOS: Cannot duplicate packet in netbios_name_cache_proxy
```

debug packet

Use the **debug packet** EXEC command to display information on packets that the network is unable to classify. The **no** form of this command disables debugging output.

debug packet
no debug packet

Syntax Description

This command has no arguments or keywords.

Command Mode

EXEC

Sample Display

Figure 1-59 shows sample **debug packet** output. Notice how similar it is to **debug broadcast** output.

```
router# debug packet

Ethernet0: Unknown ARPA, src 0000.0c00.6fa4, dst ffff.ffff.ffff, type 0x0a0
data 00000c00f23a00000c00ab45, len 60
Serial3: Unknown HDLC, size 64, type 0xaaaa, flags 0x0F00
Serial2: Unknown PPP, size 128
Serial7: Unknown FRAME-RELAY, size 174, type 0x5865, DLCI 7a
Serial0: compressed TCP/IP packet dropped
```

52692

Figure 1-59 Sample Debug Packet Output

Table 1-37 describes significant fields shown in Figure 1-59.

Table 1-37 Debug Packet Field Descriptions

Field	Description
Ethernet0	Name of the Ethernet interface that received the packet.
Unknown	States that the network was unable to classify this packet. Examples include packets with unknown link types.
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

Field	Description
	FDDI (MCM) *Encapsulation Style* APOLLO ISO1 ISO3 LLC2 SNAP
	Frame Relay *Encapsulation Style* BRIDGE FRAME-RELAY
	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
	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	MAC address of the destination node for the packet.
type 0x0a0	Packet type.
data ...	First 12 bytes of the datagram following the MAC header.
len 60	Length of the message in bytes that the interface received from the wire.
size 64	Length of the message in bytes that the interface received from the wire. Equivalent to the len field.

Field	Description
flags 0x0F00	HDLC or PP flags field.
DLCI 7a	The DLCI number on Frame Relay.
compressed TCP/IP packet dropped	This message can occur when TCP header compression is enabled on an interface and the packet does not turn out to be HDLC or X25 after classification.

debug ppp

Use the **debug ppp** EXEC command to display information on traffic and exchanges in an internetwork implementing the Point-to-Point Protocol (PPP). The **no** form of this command disables debugging output.

debug ppp {**packet** | **negotiation** | **error** | **chap**}
no debug ppp {**packet** | **negotiation** | **error** | **chap**}

Syntax Description

packet	Causes the **debug ppp** command to display PPP packets being sent and received. (This command displays low-level packet dumps.)
negotiation	Causes the **debug ppp** command to display PPP packets transmitted during PPP startup, where PPP options are negotiated
error	Causes the **debug ppp** command to display protocol errors and error statistics associated with PPP connection negotiation and operation.
chap	Causes the **debug ppp** command to display Challenge Authentication Protocol (CHAP) packet exchanges.

Command Mode

EXEC

Usage Guidelines

Use the **debug ppp** commands when trying to find the following:

- The Network Control Protocols (NCPs) that are supported on either end of a PPP connection
- Any loops that might exist in a PPP internetwork
- Nodes that are (or are not) properly negotiating PPP connections
- Errors that have occurred over the PPP connection
- Causes for CHAP session failures

Refer to Internet RFCs 1331, 1332, and 1333 for details concerning PPP-related nomenclature and protocol information.

Sample Display

Figure 1-60 provides a sample **debug ppp packet** output as seen from the Link Quality Monitor (LQM) side of the connection. This display example depicts packet exchanges under normal PPP operation.

```
router# debug ppp packet

PPP Serial4(o): lcp_slqr() state = OPEN magic = D21B4, len = 48
PPP Serial4(i): pkt type 0xC025, datagramsize 52
PPP Serial4(i): lcp_rlqr() state = OPEN magic = D3454, len = 48
PPP Serial4(i): pkt type 0xC021, datagramsize 16
PPP Serial4: I LCP ECHOREQ(9) id 3 (C) magic D3454
PPP Serial4: input(C021) state = OPEN code = ECHOREQ(9) id = 3 len = 12
PPP Serial4: O LCP ECHOREP(A) id 3 (C) magic D21B4
PPP Serial4(o): lcp_slqr() state = OPEN magic = D21B4, len = 48
PPP Serial4(i): pkt type 0xC025, datagramsize 52
PPP Serial4(i): lcp_rlqr() state = OPEN magic = D3454, len = 48
PPP Serial4(i): pkt type 0xC021, datagramsize 16
PPP Serial4: I LCP ECHOREQ(9) id 4 (C) magic D3454
PPP Serial4: input(C021) state = OPEN code = ECHOREQ(9) id = 4 len = 12
PPP Serial4: O LCP ECHOREP(A) id 4 (C) magic D21B4
PPP Serial4(o): lcp_slqr() state = OPEN magic = D21B4, len = 48
PPP Serial4(i): pkt type 0xC025, datagramsize 52
PPP Serial4(i): lcp_rlqr() state = OPEN magic = D3454, len = 48
PPP Serial4(i): pkt type 0xC021, datagramsize 16
PPP Serial4: I LCP ECHOREQ(9) id 5 (C) magic D3454
PPP Serial4: input(C021) state = OPEN code = ECHOREQ(9) id = 5 len = 12
PPP Serial4: O LCP ECHOREP(A) id 5 (C) magic D21B4
PPP Serial4(o): lcp_slqr() state = OPEN magic = D21B4, len = 48
PPP Serial4(i): pkt type 0xC025, datagramsize 52
PPP Serial4(i): lcp_rlqr() state = OPEN magic = D3454, len = 48
PPP Serial4(i): pkt type 0xC021, datagramsize 16
PPP Serial4: I LCP ECHOREQ(9) id 6 (C) magic D3454
PPP Serial4: input(C021) state = OPEN code = ECHOREQ(9) id = 6 len = 12
PPP Serial4: O LCP ECHOREP(A) id 6 (C) magic D21B4
PPP Serial4(o): lcp_slqr() state = OPEN magic = D21B4, len = 48
PPP Serial4(i): pkt type 0xC025, datagramsize 52
PPP Serial4(i): lcp_rlqr() state = OPEN magic = D3454, len = 48
PPP Serial4(i): pkt type 0xC021, datagramsize 16
PPP Serial4: I LCP ECHOREQ(9) id 7 (C) magic D3454
PPP Serial4: input(C021) state = OPEN code = ECHOREQ(9) id = 7 len = 12
PPP Serial4: O LCP ECHOREP(A) id 7 (C) magic D21B4
PPP Serial4(o): lcp_slqr() state = OPEN magic = D21B4, len = 48
```

S2693

Figure 1-60 Sample Debug PPP Packet Display Output

Explanations for individual fields of output for the **debug ppp packet** command follow in Table 1-38.

Table 1-38 Debug PPP Packet Field Descriptions

Field	Description
PPP	Indicates that this is PPP debugging output.
Serial4	Interface number associated with this debugging information.
(o), O	Both indicate that this packet was detected as an output packet.
(i) I	Both indicate that this packet was detected as an input packet.
lcp_slqr()	Procedure name; running LQM, send a Link Quality Report (LQR).
lcp_rlqr()	Procedure name; running LQM, received an LQR.
input (C025)	Indicates that the router received a packet of the specified packet type (in hex). A value of C025 indicates packet of type LQM.
state = OPEN	PPP state; normal state is OPEN.
magic = D21B4	Magic Number for indicated node; when output is indicated, this is the Magic Number of the node on which debugging is enabled. The actual Magic Number depends on whether the packet detected is indicated as I or O.
datagramsize = 52	Packet length including header.
code = ECHOREQ(9)	Code identifies the type of packet received. Both forms of the packet, string and hexadecimal, are presented.
len = 48	Packet length without header.
id = 3	ID number per Link Control Protocol (LCP) packet format.
pkt type 0xC025	Packet type in hexadecimal; typical packet types are C025 for LQM and C021 for LCP.
LCP ECHOREQ (9)	Specifies Echo Request; value in parentheses is the hexadecimal representation of the LCP type.
LCP ECHOREP (A)	Specifies Echo Reply; value in parentheses is the hexadecimal representation of the LCP type.

To elaborate on what the router is displaying here, consider the partial exchange in Figure 1-61. This sequence shows that one side is using ECHO for its keepalives and the other side is using LQRs.

```
PPP Serial4(o): lcp_slqr() state = OPEN magic = D21B4, len = 48
PPP Serial4(i): pkt type 0xC025, datagramsize 52
PPP Serial4(i): lcp_rlqr() state = OPEN magic = D3454, len = 48
PPP Serial4(i): pkt type 0xC021, datagramsize 16
PPP Serial4: I LCP ECHOREQ(9) id 3 (C) magic D3454
PPP Serial4: input(C021) state = OPEN code = ECHOREQ(9) id = 3 len = 12
PPP Serial4: O LCP ECHOREP(A) id 3 (C) magic D21B4
PPP Serial4(o): lcp_slqr() state = OPEN magic = D21B4, len = 48
```

S2694

Figure 1-61 Partial Debug PPP Packet Display Output

The following discussion briefly outlines each line of this exchange.

The first line states that the router with debugging enabled has sent an LQR to the other side of the PPP connection:

```
PPP Serial4(o): lcp_slqr() state = OPEN magic = D21B4, len = 48
```

The next two lines indicate that the router has received a packet of type C025 (LQM) and provides details about the packet:

```
PPP Serial4(i): pkt type 0xC025, datagramsize 52
PPP Serial4(i): lcp_rlqr() state = OPEN magic = D3454, len = 48
```

The next two lines indicate that the router received an ECHOREQ of type C021 (LCP). The other side is sending ECHOs. The router on which debugging is configured for LQM but also responds to ECHOs.

```
PPP Serial4(i): pkt type 0xC021, datagramsize 16
PPP Serial4: I LCP ECHOREQ(9) id 3 (C) magic D3454
```

Next the router is detected to have responded to the ECHOREQ with an ECHOREP and is preparing to send out an LQR:

```
PPP Serial4: O LCP ECHOREP(A) id 3 (C) magic D21B4
PPP Serial4(o): lcp_slqr() state = OPEN magic = D21B4, len = 48
```

Figure 1-62 provides a sample **debug ppp negotiation** output. This is a normal negotiation, where both sides agree on NCP parameters. In this case, protocol type IP is proposed and acknowledged.

```
router# debug ppp negotiation

ppp: sending CONFREQ, type = 4 (CI_QUALITYTYPE), value = C025/3E8
ppp: sending CONFREQ, type = 5 (CI_MAGICNUMBER), value = 3D56CAC
ppp: received config for type = 4 (QUALITYTYPE) acked
ppp: received config for type = 5 (MAGICNUMBER) value = 3D567F8 acked (ok)
PPP Serial4: state = ACKSENT fsm_rconfack(C021): rcvd id 5
ppp: config ACK received, type = 4 (CI_QUALITYTYPE), value = C025
ppp: config ACK received, type = 5 (CI_MAGICNUMBER), value = 3D56CAC
ppp: ipcp_reqci: returning CONFACK.
 (ok)
PPP Serial4: state = ACKSENT fsm_rconfack(8021): rcvd id 4
```

S2695

Figure 1-62 Sample Debug PPP Negotiation Display Output

Explanations for key individual fields of output from the **debug ppp negotiation** command follow in Table 1-39.

Table 1-39 Debug PPP Negotiation Field Descriptions

Field	Description
ppp	Indicates that this is a PPP debugging output.
sending CONFREQ	Indicates that the router sent a configuration request.
type = 4 (CI_QUALITYTYPE)	Specifies the type of LCP configuration option that is being negotiated and a descriptor. A type value of 4 indicates Quality Protocol negotiation; a type value of 5 indicates Magic Number negotiation.
value = C025/3E8	For Quality Protocol negotiation, indicates NCP type and reporting period. In the example, C025 indicates LQM; 3E8 is a hexadecimal value translating to about 10 seconds (in hundredths of a second).
value = 3D56CAC	For Magic Number negotiation, indicates the Magic Number being negotiated.
received config	Indicates that the receiving node has received the proposed option negotiation for the indicated option type.
acked	Indicates acknowledgment and acceptance of options.
state = ACKSENT	Indicates the specific PPP state in the negotiation process.
ipcp_reqci	IPCP notification message; sending CONFACK
fsm_rconfack (8021)	The procedure fsm_rconfack processes received CONFACKs, and the protocol (8021) is IP.

The following discussion briefly outlines each line shown in the example provided in Figure 1-62.

The first two lines in Figure 1-62 indicate that the router is trying to bring up LCP and intends to use the indicated negotiation options (Quality Protocol and Magic Number). The value fields are the values of the options themselves. C025/3E8 translates to Quality Protocol LQM. 3E8 is the reporting period (in hundredths of a second). 3D56CAC is the value of the Magic Number for the router.

```
ppp: sending CONFREQ, type = 4 (CI_QUALITYTYPE), value = C025/3E8
ppp: sending CONFREQ, type = 5 (CI_MAGICNUMBER), value = 3D56CAC
```

The next two lines indicate that the other side negotiated for options 4 and 5 as requested and acknowledged both. If the responding end does not support the options, a CONFREJ is sent by the responding node. If the responding end does not like the value of the option, a CONFNAK is sent with the value field modified.

```
ppp: received config for type = 4 (QUALITYTYPE) acked
ppp: received config for type = 5 (MAGICNUMBER) value = 3D567F8 acked (ok)
```

The next three messages indicate that the router received a CONFACK from the responding side and displays accepted option values. Use the rcvd id field to verify the CONFREQ and CONFACK have the same id field.

```
PPP Serial4: state = ACKSENT fsm_rconfack(C021): rcvd id 5
ppp: config ACK received, type = 4 (CI_QUALITYTYPE), value = C025
ppp: config ACK received, type = 5 (CI_MAGICNUMBER), value = 3D56CAC
```

The next **debug ppp negotiation** command output indicates that the router has IP routing enabled on this interface and that the IPCP NCP negotiated successfully.

```
ppp: ipcp_reqci: returning CONFACK.
```

In the last message, the router's state is listed as ACKSENT:

```
PPP Serial4: state = ACKSENT fsm_rconfack(C021): rcvd id 5\
```

Figure 1-63 provides a sample display output when both **debug ppp packet** and **debug ppp negotiation** output are enabled at the same time.

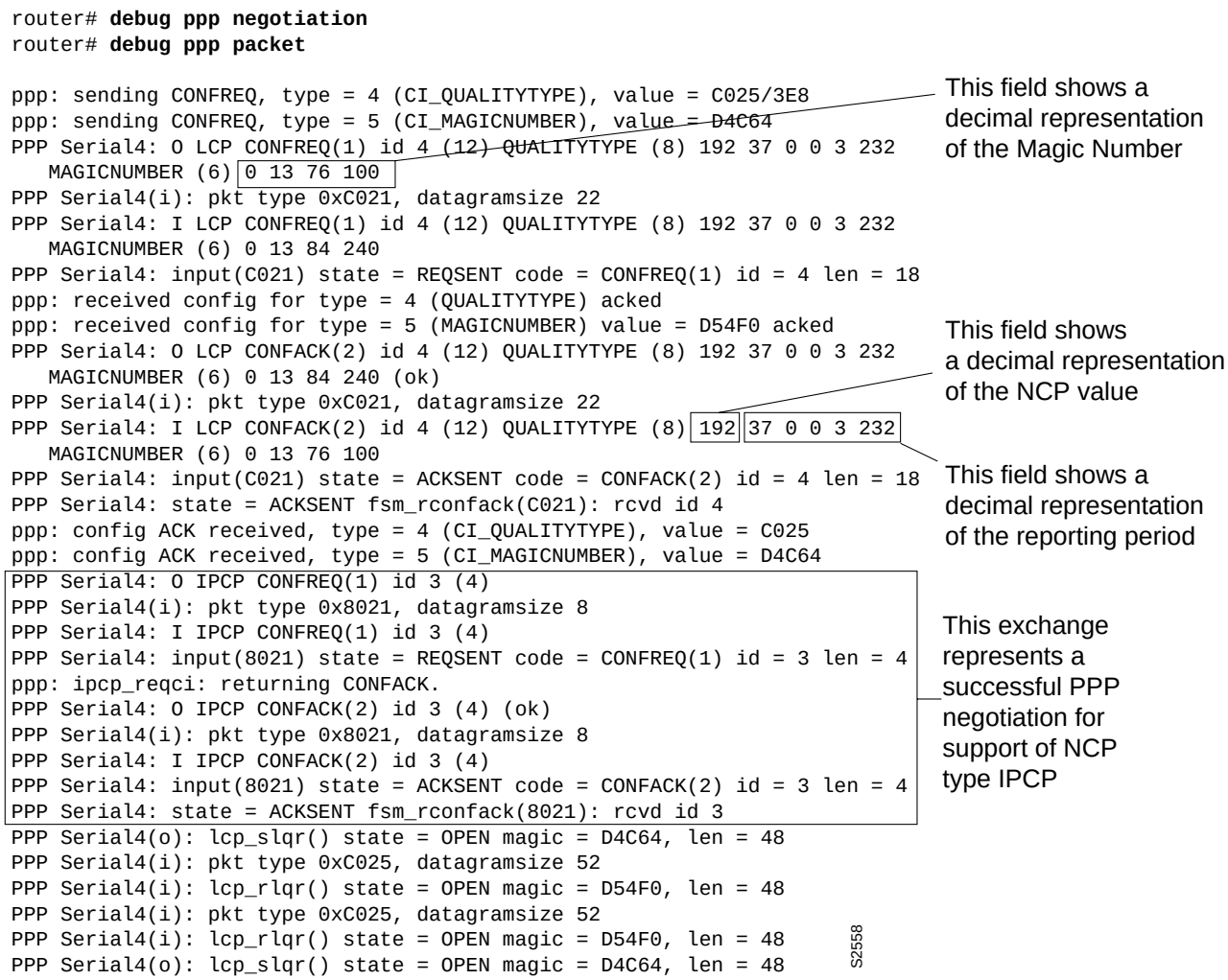


Figure 1-63 Sample Debug PPP Display Output with Both Options Enabled

Figure 1-64 provides a sample **debug ppp negotiation** display output when the remote side of the connection is unable to respond to LQM requests.

```
router# debug ppp negotiation

ppp: sending CONFREQ, type = 4 (CI_QUALITYTYPE), value = C025/3E8
ppp: sending CONFREQ, type = 5 (CI_MAGICNUMBER), value = 44B7010
ppp: sending CONFREQ, type = 4 (CI_QUALITYTYPE), value = C025/3E8
ppp: sending CONFREQ, type = 5 (CI_MAGICNUMBER), value = 44B7010
ppp: sending CONFREQ, type = 4 (CI_QUALITYTYPE), value = C025/3E8
ppp: sending CONFREQ, type = 5 (CI_MAGICNUMBER), value = 44B7010
ppp: sending CONFREQ, type = 4 (CI_QUALITYTYPE), value = C025/3E8
ppp: sending CONFREQ, type = 5 (CI_MAGICNUMBER), value = 44B7010
ppp: sending CONFREQ, type = 4 (CI_QUALITYTYPE), value = C025/3E8
ppp: sending CONFREQ, type = 5 (CI_MAGICNUMBER), value = 44B7010
ppp: sending CONFREQ, type = 4 (CI_QUALITYTYPE), value = C025/3E8
ppp: sending CONFREQ, type = 5 (CI_MAGICNUMBER), value = 44B7010
ppp: sending CONFREQ, type = 4 (CI_QUALITYTYPE), value = C025/3E8
ppp: sending CONFREQ, type = 5 (CI_MAGICNUMBER), value = 44B7010
ppp: sending CONFREQ, type = 4 (CI_QUALITYTYPE), value = C025/3E8
ppp: sending CONFREQ, type = 5 (CI_MAGICNUMBER), value = 44B7010
ppp: sending CONFREQ, type = 4 (CI_QUALITYTYPE), value = C025/3E8
ppp: sending CONFREQ, type = 5 (CI_MAGICNUMBER), value = 44B7010
ppp: sending CONFREQ, type = 4 (CI_QUALITYTYPE), value = C025/3E8
ppp: sending CONFREQ, type = 5 (CI_MAGICNUMBER), value = 44B7010
ppp: sending CONFREQ, type = 4 (CI_QUALITYTYPE), value = C025/3E8
ppp: sending CONFREQ, type = 5 (CI_MAGICNUMBER), value = 44B7010
ppp: sending CONFREQ, type = 4 (CI_QUALITYTYPE), value = C025/3E8
ppp: sending CONFREQ, type = 5 (CI_MAGICNUMBER), value = 44C1488
```

S2696

Figure 1-64 Sample Debug PPP Negotiation Display Output when No Response Is Detected

Figure 1-65 provides a sample display output when no response is detected for configuration requests (with both **debug ppp negotiation** and **debug ppp packet** enabled).

```
router# debug ppp packet

ppp: sending CONFREQ, type = 4 (CI_QUALITYTYPE), value = C025/3E8
ppp: sending CONFREQ, type = 5 (CI_MAGICNUMBER), value = 44DFDC8
PPP Serial4: O LCP CONFREQ(1) id 14 (12) QUALITYTYPE (8) 192 37 0 0 3 232
   MAGICNUMBER (6) 4 77 253 200
ppp: TIMEout: Time= 44E0980 State= 3
ppp: sending CONFREQ, type = 4 (CI_QUALITYTYPE), value = C025/3E8
ppp: sending CONFREQ, type = 5 (CI_MAGICNUMBER), value = 44DFDC8
PPP Serial4: O LCP CONFREQ(1) id 15 (12) QUALITYTYPE (8) 192 37 0 0 3 232
   MAGICNUMBER (6) 4 77 253 200
ppp: TIMEout: Time= 44E1828 State= 3
ppp: sending CONFREQ, type = 4 (CI_QUALITYTYPE), value = C025/3E8
ppp: sending CONFREQ, type = 5 (CI_MAGICNUMBER), value = 44DFDC8
PPP Serial4: O LCP CONFREQ(1) id 16 (12) QUALITYTYPE (8) 192 37 0 0 3 232
   MAGICNUMBER (6) 4 77 253 200
ppp: TIMEout: Time= 44E27C8 State= 3
ppp: sending CONFREQ, type = 4 (CI_QUALITYTYPE), value = C025/3E8
ppp: sending CONFREQ, type = 5 (CI_MAGICNUMBER), value = 44DFDC8
PPP Serial4: O LCP CONFREQ(1) id 17 (12) QUALITYTYPE (8) 192 37 0 0 3 232
   MAGICNUMBER (6) 4 77 253 200
ppp: TIMEout: Time= 44E3768 State= 3
```

S2697

Figure 1-65 Sample Debug PPP Display Output when No Response Is Detected (with Both Options Enabled)

Figure 1-66 provides a sample **debug ppp error** output. These messages might appear when the Quality Protocol option is enabled on an interface that is already running PPP.

```
router# debug ppp error

PPP Serial3(i): rlqr receive failure.  successes = 15
PPP: myrcvdiffp = 159 peerxmitdiffp = 41091
PPP: myrcvdiffo = 2183 peerxmitdiffo = 1714439
PPP: threshold = 25
PPP Serial4(i): rlqr transmit failure. successes = 15
PPP: myxmitdiffp = 41091 peerrcvdiffp = 159
PPP: myxmitdiffo = 1714439 peerrcvdiffo = 2183
PPP: l->OutLQRs = 1 LastOutLQRs = 1
PPP: threshold = 25
PPP Serial3(i): lqr_protrej() Stop sending LQRs.
PPP Serial3(i): The link appears to be looped back.
```

S2698

Figure 1-66 Sample Debug PPP Error Output

Explanations for individual fields of output from **debug ppp errors** follow in Table 1-40.

Table 1-40 Debug PPP Error Field Descriptions

Field	Description
PPP	Indicates that this is PPP debugging output.
Serial3(i)	Interface number associated with this debugging information; indicates that this is an input packet.
rlqr receive failure	Indicates that the request to negotiate the Quality Protocol option is not accepted.
myrcvdiffp = 159	Number of packets received over the time period.
peerxmitdiffp = 41091	Number of packets sent by the remote node over this period.
myrcvdiffo = 2183	Number of octets received over this period.
peerxmitdiffo = 1714439	Number of octets sent by the remote node over this period.
threshold = 25	The maximum error percentage acceptable on this interface. This percentage is calculated by the threshold value entered in the **ppp quality** *number* interface configuration command. A value of 100–*number* (100 minus *number*) is the maximum error percentage. In this case, a *number* of 75 was entered. This means that the local router must maintain a minimum 75 percent non-error percentage, or the PPP link will be considered down.
OutLQRs = 1	Local router's current send LQR sequence number.
LastOutLQRs = 1	The last sequence number that the remote node side has seen from the local node.

Figure 1-67 provides a sample **debug ppp chap** output. When doing CHAP authentication, use this **debug** command to determine why an authentication fails.

```
router# debug ppp chap

Serial0: Unable to authenticate.  No name received from peer
Serial0: Unable to validate CHAP response.  USERNAME pioneer not found.
Serial0: Unable to validate CHAP response.  No password defined for USERNAME pioneer
Serial0: Failed CHAP authentication with remote.
Remote message is Unknown name
Serial0: remote passed CHAP authentication.
Serial0: Passed CHAP authentication with remote.
Serial0: CHAP input code = 4 id = 3 len = 48
```

S2699

Figure 1-67 Sample Debug PPP CHAP Output

In general, these messages are self-explanatory. Fields that appear in **debug ppp chap** displays that can show optional output are outlined in Table 1-41.

Table 1-41 Debug PPP CHAP Field Descriptions

Field	Description
Serial0	Interface number associated with this debugging information and CHAP access session in question.
USERNAME pioneer not found.	The name *pioneer* in this example is the name received in the CHAP response. The router looks up this name in the list of usernames that are configured for the router.
Remote message is Unknown name	Messages that can appear are the following: No name received to authenticate Unknown name No secret for given name Short MD5 response received MD compare failed
code = 4	Specific CHAP type packet detected. Possible values are as follows: 1 = Challenge 2 = Response 3 = Success 4 = Failure
len = 48	Packet length without header.
id = 3	ID number per Link Control Protocol (LCP) packet format.

debug rif

Use the **debug rif** EXEC command to display information on entries entering and leaving the RIF cache. The **no** form of this command disables debugging output.

debug rif
no debug rif

Syntax Description

This command has no arguments or keywords.

Command Mode

EXEC

Usage Guidelines

In order to use the **debug rif** command to display traffic source-routed through an interface, fast switching of SRB frames must first be disabled with the **no source-bridge route-cache** interface interface configuration command.

Sample Display

Figure 1-68 shows sample **debug rif** output.

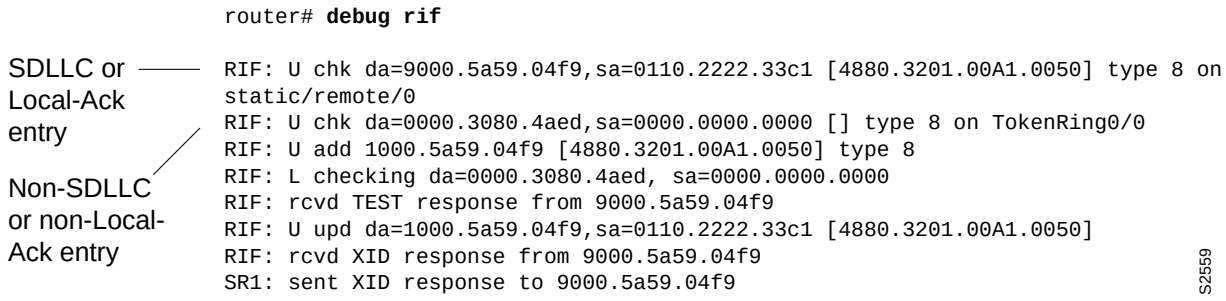


Figure 1-68 Sample Debug RIF Output

Explanations for representative lines of **debug rif** output in Figure 1-68 follow.

The first line of output in Figure 1-68 is an example of a RIF entry for an interface configured for SDLLC or Local-Ack.

Table 1-42 describes significant fields shown in this line of **debug rif** output.

Table 1-42 Debug RIF Field Descriptions—Part 1

Field	Description
RIF:	Indicates that this message describes RIF debugging output.
U chk	Update checking. The entry is being updated; the timer is set to zero (0).
da = 9000.5a59.04f9	Destination MAC address.
sa = 0110.2222.33c1	Source MAC address. This field contains values of zero (0000.0000.0000) in a non-SDLLC or non-Local-ack entry.
[4880.3201.00A1.0050]	RIF string. This field is blank (null RIF) in a non-SDLLC or non-Local-ack entry.
type 8	Possible values follow: 0—Null entry 1—This entry was learned from a particular Token Ring port (interface) 2—Statically configured 4—Statically configured for a remote interface 8—This entry is to be aged 16—This entry (which has been learned from a remote interface) is to be aged 32—This entry is not to be aged 64 —This interface is to be used by LAN Network Manager (and is not to be aged)
on static/remote/0	Indicates that this route was learned from a real Token Ring port, in contrast to a virtual ring.

The second line of output in Figure 1-68 is an example of a RIF entry for an interface that is not configured for SDLLC or Local-ACK.

```
RIF: U chk da=0000.3080.4aed,sa=0000.0000.0000 [] type 8 on TokenRing0/0
```

Notice that the source address contains only zero values (0000.0000.0000), and that the RIF string is null ([ ]). The last element in the entry indicates that this route was learned from a virtual ring, rather than a real Token Ring port.

The third line of output in Figure 1-68 shows that a new entry has been added to the RIF cache:

```
RIF: U add 1000.5a59.04f9 [4880.3201.00A1.0050] type 8
```

The fourth line of output in Figure 1-68 shows that a RIF cache lookup operation has taken place:

```
RIF: L checking da=0000.3080.4aed, sa=0000.0000.0000
```

The fifth line of output in Figure 1-68 shows that a TEST response from address 9000.5a59.04f9 was inserted into the RIF cache:

```
RIF: rcvd TEST response from 9000.5a59.04f9
```

The sixth line of output in Figure 1-68 shows that the RIF entry for this route has been found and updated:

```
RIF: U upd da=1000.5a59.04f9,sa=0110.2222.33c1 [4880.3201.00A1.0050]
```

The seventh line of output in Figure 1-68 shows that an XID response from this address was inserted into the RIF cache:

```
RIF: rcvd XID response from 9000.5a59.04f9
```

The eighth line of output in Figure 1-68 shows that the router sent an XID response to this address:

```
SR1: sent XID response to 9000.5a59.04f9
```

Table 1-43 explains the other possible lines of **debug rif** output.

Table 1-43 Debug RIF Field Descriptions—Part 2

Field	Description
RIF: L Sending XID for *address*	The router/bridge wanted to send a packet to *address* but did not find it in the RIF cache. It sent an XID explorer packet to determine which RIF it should use. The attempted packet is dropped.
RIF: L No buffer for XID to *address*	Similar to the previous description; however, a buffer in which to build the XID packet could not be obtained.
RIF: U remote rif too small [*rif*]	A packet's RIF was too short to be valid.
RIF: U rej *address* too big [*rif*]	A packet's RIF exceeded the maximum size allowed and was rejected. The maximum size is 18 bytes.
RIF: U upd interface *address*	The RIF entry for this router/bridge's interface has been updated.
RIF: U ign *address* interface update	A RIF entry that would have updated an interface corresponding to one of this router's interfaces.
RIF: U add *address*[*rif*]	The RIF entry for *address* has been added to the RIF cache.
RIF: U no memory to add rif for *address*	No memory to add a RIF entry for *address*.
RIF: removing rif entry for *address*, *type code*	The RIF entry for *address* has been forcibly removed.
RIF: flushed *address*	The RIF entry for *address* has been removed because of a RIF cache flush.
RIF: expired *address*	The RIF entry for *address* has been aged out of the RIF cache.

debug sdlc

Use the **debug sdlc** EXEC command to display information on SDLC frames received and sent by any router serial interface involved in supporting SDLC end station functions. The **no** form of this command disables debugging output.

debug sdlc
no debug sdlc

Syntax Description

This command has no arguments or keywords.

Command Mode

EXEC

Usage Guidelines

Because using this command is processor intensive, it is best to use it after hours, rather than in a production environment. It is also best to turn this command on by itself, rather than use it in conjunction with other **debug** commands.

Sample Display

Figure 1-69 shows sample **debug sdlc** output.

```
router# debug sdlc

SDLC: Sending RR at location 4
Serial3: SDLC O (12495952) C2 CONNECT (2) RR P/F 6
Serial3: SDLC I (12495964) [C2] CONNECT (2) RR P/F 0 (R) [VR: 6 VS: 0]
Serial3: SDLC T [C2] 12496064 CONNECT 12496064 0
SDLC: Sending RR at location 4
Serial3: SDLC O (12496064) C2 CONNECT (2) RR P/F 6
Serial3: SDLC I (12496076) [C2] CONNECT (2) RR P/F 0 (R) [VR: 6 VS: 0]
Serial3: SDLC T [C2] 12496176 CONNECT 12496176 0
```

S2700

Figure 1-69 Sample Debug SDLC Output

Explanations for individual lines of output from Figure 1-69 follow.

The following line of output indicates that the router is sending a Receiver Ready packet at location 4 in the code:

```
SDLC: Sending RR at location 4
```

The following line of output describes a frame input event:

```
Serial3: SDLC O (12495952) C2 CONNECT (2) RR P/F 6
```

Table 1-44 describes the fields in this line of output.

Table 1-44 Debug SDLC Field Descriptions for a Frame Output Event

Field	Description
Serial3	Interface type and unit number reporting the frame event.
SDLC	Protocol providing the information.
O	Command Mode of frame event. Possible values follow: I—Frame input O—Frame output T—T1 timer expired
(12495952)	Current timer value.
C2	SDLC address of the SDLC connection.
CONNECT	State of the protocol when the frame event occurred. Possible values follow: CONNECT DISCONNECT DISCSENT (disconnect sent) ERROR (FRMR frame sent) REJSENT (reject frame sent) SNRMSENT (SNRM frame sent) USBUSY THEMBUSY BOTHBUSY
(2)	Size of the frame (in bytes).

Field	Description
RR	Frame type name. Possible values follow: DISC—Disconnect DM—Disconnect mode FRMR—Frame reject IFRAME—Information frame REJ—Reject RNR—Receiver not ready RR—Receiver ready SIM—Set Initialization mode command SNRM—Set Normal Response Mode TEST—Test frame UA—Unnumbered acknowledgment XID—EXchange ID
P/F	Poll/Final bit indicator. Possible values follow: F—Final (printed for Response frames) P—Poll (printed for Command frames) P/F—Poll/Final (printed for RR, RNR and REJ frames, which can be either Command or Response frames)
6	Receive count; range: 0–7.

The following line of output describes a frame input event.

```
Serial3: SDLC I (12495964) [C2] CONNECT (2) RR P/F 0 (R) [VR: 6 VS: 0] rfp: P
```

In addition to the fields described in Table 1-44, output for a frame input event also includes two additional fields, as described in Table 1-45.

Table 1-45 Debug SDLC Field Descriptions Unique to a Frame Input Event

Field	Description
(R)	Frame Type: C—Command R—Response
VR: 6	Receive count; range: 0–7.
VS: 0	Send count; range: 0–7.
rfp: P	Ready for poll; P —Idle poll (keepalive) timer is on. T—Data acknowledgment timer is on. These timers are based on the T1 timer.
VS: 0	Send count; range: 0–7.

The following line of output describes a frame timer event.

```
Serial3: SDLC T [C2] 12496064 CONNECT 12496064 0
```

Table 1-46 describes the fields in this line of output.

Table 1-46 Debug SDLC Field Descriptions for a Timer Event

Field	Description
Serial3:	Interface type and unit number reporting the frame event.
SDLC	Protocol providing the information.
T	Indicates that the timer has expired.
[C2]	SDLC address of this SDLC connection.
12496064	System clock.
CONNECT	State of the protocol when the frame event occurred. Possible values follow: BOTHBUSY CONNECT DISCONNECT DISCSENT (disconnect sent) ERROR (FRMR frame sent) REJSENT (reject frame sent) SNRMSENT (SNRM frame sent) THEMBUSY BOTHBUSY
12496064	Top timer.
0	Retry count; default: 0.

debug sdlc local-ack

Use the **debug sdlc local-ack** EXEC command to display information on the Local Acknowledgment feature. The **no** form of this command disables debugging output.

debug sdlc local-ack
no debug sdlc local-ack

Syntax Description

This command has no arguments or keywords.

Command Mode

EXEC

Usage Guidelines

You can select the frame types you want to monitor; the frame types correspond to bit flags. You can select 1, 2, 4, or 7, which is the decimal value of the bit flag settings. If you select 1, the octet is set to 00000001. If you select 2, the octet is set to 0000010. If you select 4, the octet is set to 00000100. If you want to select all frame types, select 7; the octet is 00000111. The default is 7 for all events. Table 1-47 defines these bit flags.

Table 1-47 Debug SDLC Local-Ack Debugging Levels

Debug Command	Meaning
debug sdlc local-ack 1	Only U-Frame events
debug sdlc local-ack 2	Only I-Frame events
debug sdlc local-ack 4	Only S-Frame events
debug sdlc local-ack 7	All SDLC Local-Ack events (default setting)

Caution Because using this command is processor intensive, it is best to use it after hours, rather than in a production environment. It is also best to turn this command on by itself, rather than use it in conjunction with other debug commands.

Sample Display

Figure 1-70 shows sample **debug sdlc local-ack** output.

```
router# debug sdlc local-ack 1

SLACK (Serial3): Input     = Network, LinkupRequest
SLACK (Serial3): Old State = AwaitSdlcOpen           New State = AwaitSdlcOpen

SLACK (Serial3): Output    = SDLC, SNRM

SLACK (Serial3): Input     = SDLC, UA
SLACK (Serial3): Old State = AwaitSdlcOpen           New State = Active

SLACK (Serial3): Output    = Network, LinkResponse
```

Group of associated operations

S2560

Figure 1-70 Sample Debug SDLC Local-Ack Output

Explanations for individual lines of output from Figure 1-70 follow.

The first line of output in the first group of lines shows the input to the SDLC Local Acknowledgment state machine:

```
SLACK (Serial3): Input     = Network, LinkupRequest
```

Table 1-48 describes the fields in this line of output.

Table 1-48 Debug SDLC Local-Ack Field Descriptions

Field	Description
SLACK	Indicates that the SDLC Local-Acknowledgment feature is providing the information.
(Serial3):	Interface type and unit number reporting the event.
Input = Network	Indicates that the source of the input.
LinkupRequest	Indicates the op code. A LinkupRequest is an example of possible values.

The second line of output shows the change in the SDLC Local Acknowledgment state machine. In this case the AwaitSdlcOpen state is an internal state that has not changed while this display was captured.

```
SLACK (Serial3): Old State = AwaitSdlcOpen            New State = AwaitSdlcOpen
```

The third line of output shows the output from the SDLC Local Acknowledgment state machine:

```
SLACK (Serial3): Input     = Network, LinkupRequest
```

debug sdllc

Use the **debug sdllc** EXEC command to display information about data link layer frames transferred between a device on a Token Ring and a device on a serial line via a router configured with the SDLLC feature. The **no** form of this command disables debugging output.

debug sdllc
no debug sdllc

Syntax Description

This command has no arguments or keywords.

Command Mode

EXEC

Usage Guidelines

The SDLLC feature translates between the SDLC link layer protocol used to communicate with devices on a serial line and the LLC2 link layer protocol used to communicate with devices on a Token Ring.

The router configured with the SDLLC feature must be attached to the serial line. The router sends and receives frames on behalf of the serial device on the attached serial line but acts as an SDLC station.

The topology between the router configured with the SDLLC feature and the Token Ring is network dependent and is not limited by the SDLLC feature.

Sample Display

Figure 1-71 shows sample **debug sdllc** output between link layer peers from the perspective of the SDLLC-configured router.

```
router# debug sdllc

SDLLC: rx explorer rsp, da 4000.2000.1001, sa C000.1020.1000, rif
 8840.0011.00A1.0050
SDLLC: tx short xid, sa 4000.2000.1001, da C000.1020.1000, rif
 88C0.0011.00A1.0050, dsap 4 ssap 4
SDLLC: tx long xid, sa 4000.2000.1001, da C000.1020.1000, rif
 88C0.0011.00A1.0050, dsap 4 ssap 4
Rcvd SABME/LINKUP_REQ pak from TR host
```

S2701

Figure 1-71 Sample Debug SDLLC Output

Explanations for individual fields of output from **debug sdllc** follow in Table 1-49:

Table 1-49 Debug SDLLC Field Descriptions

Field	Description
rx	Router receives message from the FEP.
explr rsp	Response to an explorer (TEST) frame previously sent by the router to FEP.
da	Destination address. This is the address of the router receiving the response.
sa	Source address. This is the address of the FEP sending the response to the router.
rif	Routing information field.
tx	Router sent message to the FEP.
short xid	Router sent the null XID to the FEP.
dsap	Destination service access point
ssap	Source service access point.
tx long xid	Router sent the XID type 2 to the FEP.
Rcvd	Router received layer 2 message from the FEP
SABME/LINKUP_REQ	Set asynchronous Balanced Mode Extended command.

The following line of output indicates that an explorer frame response has been received by the router at address 4000.2000.1001 from the FEP at address C000.1020.1000 with the specified RIF. The original explorer sent to the FEP from the router is not monitored as part of the **debug sdllc** command.

```
SDLLC: rx explorer rsp, da 4000.2000.1001, sa C000.1020.1000, rif
 8840.0011.00A1.0050
```

The following line of output indicates that the router sent the null XID (Type 0) to the FEP. The debugging information does not include the response to the XID message sent by the FEP to the router.

```
SDLLC: tx short xid, sa 4000.2000.1001, da C000.1020.1000, rif
 88C0.0011.00A1.0050, dsap 4 ssap 4
```

The following line of output indicates that the router sent the XID command (Format 0 Type 2) to the FEP:

```
SDLLC: tx long xid, sa 4000.2000.1001, da C000.1020.1000, rif
 88C0.0011.00A1.0050, dsap 4 ssap 4
```

The following line of output is the SABME response to the XID command previously sent by the router to the FEP:

```
Rcvd SABME/LINKUP_REQ pak from TR host
```

debug serial interface

Use the **debug serial interface** EXEC command to display information on a serial connection failure. The **no** form of this command disables debugging output.

debug serial interface
no debug serial interface

Syntax Description

This command has no arguments or keywords.

Command Mode

EXEC

Usage Guidelines

If the **show interface serial** command shows that the line and protocol are down, you can use the **debug serial interface** command to isolate a timing problem as the cause of a connection failure. If the keepalive values in the mineseq, yourseen, and myseen fields are not incrementing in each subsequent line of output, there is a timing or line problem at one of end of the connection.

Note While the **debug serial interface** command typically does not generate a lot of output, nevertheless use it cautiously during production hours. When SMDS is enabled, for example, it can generate considerable output.

The output of the **debug serial interface** command can vary, depending on the type of WAN configured for an interface: DDR, Frame Relay, HDLC, HSSI, SMDS, or X.25. The output also can vary depending on the type of encapsulation configured for that interface. The hardware platform also can impact **debug serial interface** output.

The following sections show example **debug serial interface** displays for various configurations and describe the possible output the command can generate for these configurations.

Debug Serial Interface for DDR

Table 1-50 describes the error messages that the **debug serial interface** command can generate for a serial interface being used as a V.25bis dialer for dial-on-demand routing.

Table 1-50 Debug Serial Interface 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 *xxxxxxxx* variable depend on the V.25bis device with which the router is communicating.
Serial 0: No dialer string defined. Dialing cannot occur.	This message is displayed when a packet is received that should cause a call to be placed. However, there is no dialer string configured, so dialing cannot occur. This message usually indicates a configuration problem.
Serial 0: Attempting to dial *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 *xxxxxxxx* 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.

Debug Serial Interface for Frame Relay Encapsulation

The following message is displayed if the encapsulation for the interface is Frame Relay (or HDLC) and the router attempts to send a packet containing an unknown packet type.

```
Illegal serial link type code xxx
```

Debug Serial Interface for HDLC

Figure 1-72 shows sample **debug serial interface** output for an HDLC connection when keepalives have been enabled.

```
router# debug serial interface

Serial1: HDLC myseq 636119, mineseen 636119, yourseen 515032, line up
Serial1: HDLC myseq 636120, mineseen 636120, yourseen 515033, line up
Serial1: HDLC myseq 636121, mineseen 636121, yourseen 515034, line up
Serial1: HDLC myseq 636122, mineseen 636122, yourseen 515035, line up
Serial1: HDLC myseq 636123, mineseen 636123, yourseen 515036, line up
Serial1: HDLC myseq 636124, mineseen 636124, yourseen 515037, line up
Serial1: HDLC myseq 636125, mineseen 636125, yourseen 515038, line up
Serial1: HDLC myseq 636126, mineseen 636126, yourseen 515039, line up

Serial1: HDLC myseq 636127, mineseen 636127, yourseen 515040, line up
Serial1: HDLC myseq 636128, mineseen 636127, yourseen 515041, line up
Serial1: HDLC myseq 636129, mineseen 636129, yourseen 515042, line up

Serial1: HDLC myseq 636130, mineseen 636130, yourseen 515043, line up
Serial1: HDLC myseq 636131, mineseen 636130, yourseen 515044, line up
Serial1: HDLC myseq 636132, mineseen 636130, yourseen 515045, line up
Serial1: HDLC myseq 636133, mineseen 636130, yourseen 515046, line down
Serial1: HDLC myseq 636127, mineseen 636127, yourseen 515040, line up
Serial1: HDLC myseq 636128, mineseen 636127, yourseen 515041, line up
Serial1: HDLC myseq 636129, mineseen 636129, yourseen 515042, line up
```

1 missed keepalive

3 missed keepalives; line goes down and interface is reset

S2561

Figure 1-72 Sample Debug Serial Interface Output for HDLC

In Figure 1-72, the **debug serial interface** display shows that the remote router is not receiving all of the keepalives the router is sending. When the difference in the values in the myseq and mineseen fields exceeds three, the line goes down and the interface is reset.

Table 1-51 describes significant fields shown in Figure 1-72.

Table 1-51 Debug Serial Interface Field Descriptions for HDLC

Field	Description
Serial1	Interface through which the serial connection is taking place.
HDLC	Indicates that the serial connection is an HDLC connection.
myseq 636119	The myseq counter increases by one each time the router sends a keepalive packet to the remote router.
mineseen 636119	The value of the mineseen counter reflects the last myseq sequence number the remote router has acknowledged receiving from the router. The remote router stores this value in its yourseen counter and sends that value in a keepalive packet to the router.
yourseen 515032	The yourseen counter reflects the value of the myseq sequence number the router has received in a keepalive packet from the remote router.
line up	Indicates that the connection between the routers is maintained. Value changes to line down if the values of the myseq and myseen fields in a keepalive packet differ by more than three. Value returns to line up when the interface is reset. If the line is in loopback mode, (looped) appears after this field.

Table 1-52 describes additional error messages that the **debug serial interface** command can generate for HDLC.

Table 1-52 Debug Serial Interface Error Messages for HDLC

Field	Description
Illegal serial link type code *xxx*, PC = 0x*nnnnnn*	This message is displayed if the router attempts to send a packet containing an unknown packet type.
Illegal HDLC serial type code *xxx*, PC = 0x*nnnnn*	This message is displayed if an unknown packet type is received.
Serial 0: attempting to restart	This message is displayed periodically if the interface is down. The hardware is then reset to hopefully correct the problem.
Serial 0: Received bridge packet sent to *nnnnnnnnn*	This message is displayed if a bridge packet is received over a serial interface configured for HDLC, and bridging is not configured on that interface.

Debug Serial Interface for HSSI

On an HSSI interface, the **debug serial interface** command can generate the following additional error message:

```
HSSI0: Reset from 0xnnnnnnn
```

This message indicates that the HSSI hardware has been reset. The 0x*nnnnnnn* variable is the address of the routine requesting that the hardware be reset; this value is useful only to development engineers.

Debug Serial Interface for ISDN Basic Rate

Table 1-53 describes error messages that the **debug serial interface** command can generate for ISDN Basic Rate.

Table 1-53 Debug Serial Interface Message Descriptions for ISDN Basic Rate

Message	Description
BRI: D-chan collision	Indicates that a collision on the ISDN D channel has occurred; the software will reattempt transmission.
Received SID Loss of Frame Alignment int.	Indicates that the ISDN hardware has lost frame alignment. This usually indicates a problem with the ISDN network.
Unexpected IMP int: ipr = 0x*nn*	Indicates that the ISDN hardware received an unexpected interrupt. The 0x*nn* variable indicates the value returned by the interrupt register.
BRI(d): RX Frame Length Violation. Length = *n* BRI(d): RX Nonoctet Aligned Frame BRI(d): RX Abort Sequence BRI(d): RX CRC Error BRI(d): RX Overrun Error BRI(d): RX Carrier Detect Lost	Any of these messages may be displayed when a receive error occurs on one of the ISDN channels. The (d) indicates which channel it is on. These messages may indicate a problem with the ISDN network connection.
BRI0: Reset from 0x*nnnnnnn*	Indicates that the BRI hardware has been reset. The 0x*nnnnnnn* variable is the address of the routine that requested that the hardware be reset; it is useful only to development engineers.
BRI(d): Bad state in SCMs scm1 = *x* scm2 = *x* scm3 = *x* BRI(d): Bad state in SCONs scon1 = *x* scon2 = *x* scon3 = *x* BRI(d): Bad state ub SCR; SCR = *x*	Any of these messages may be displayed if the ISDN hardware is not in the proper state. The hardware is then reset. If the message is displayed constantly, it usually indicates a hardware problem.
BRI(d): Illegal packet encapsulation = *n*	This message is displayed if a packet is received, but the encapsulation used for the packet is not recognized. It can indicate that the interface is misconfigured.

Debug Serial Interface for an MK5025 Device

Table 1-54 describes the additional error messages that the **debug serial interface** command can generate for an MK5025 device.

Table 1-54 Debug Serial Interface Message Descriptions for an MK5025 Device

Message	Description
MK5(d): Reset from 0x*nnnnnnnn*	This message indicates that the hardware has been reset. The 0x*nnnnnnn* variable is the address of the routine that requested that the hardware be reset; it is useful only to development engineers.
MK5(d): Illegal packet encapsulation = *n*	This message is displayed if a packet is received, but the encapsulation used for the packet is not recognized. Possibly an indication that the interface is misconfigured.
MK5(d): No packet available for packet realignment	This message is displayed in cases where the serial driver attempted to get a buffer (memory) and was unable to do so.
MK5(d): Bad state in CSR0 = (*x*)	This message is displayed if the hardware is not in the proper state. The hardware is then reset. If this message is displayed constantly, it usually indicates a hardware problem.
MK5(d): New serial state = *n*	This message is displayed to indicate that the hardware has interrupted the software. It displays the state that the hardware is reporting.
MK5(d): DCD is down. MK5(d): DCD is up.	If the interrupt indicates that the state of carrier has changed, one of these messages is displayed to indicate the current state of DCD.

Debug Serial Interface for PPP Encapsulation

Figure 1-73 lists all of the messages that the **debug serial interface** command can generate when the encapsulation is set to PPP and PPP is negotiating configuration options.

```
router# debug serial interface

ppp: deccp_ackci: received bad Ack
ppp: deccp_nakci: received bad Nak
ppp: deccp_rejci: received bad Reject
ppp: ipcp_reqci: bad CI length
ppp: ipcp_ackci: received bad Ack
ppp: ipcp_nakci: received bad Nak
ppp: ipcp_rejci: received bad Reject
ppp: ipcp_reqci: bad CI length
ppp: ipcp_reqci: returning CONFACK
ppp: ipcp_reqci: returning CONFNAK
ppp: ipcp_reqci: returning CONFREJ
ppp: rcvd short code-reject packet
ppp: rcvd code-reject for code n
ppp: received bad configuration ACK
ppp: received bad configuration NAK
ppp: received bad configuration reject
ppp: bad CI length = n
ppp: rcvd unknown option n
```

S2702

Figure 1-73 Sample Debug Serial Interface Output for PPP

A knowledge of the PPP protocol is necessary to understand the significance of the messages listed in Figure 1-73.

Figure 1-74 lists the **debug serial interface** messages that can be displayed when CHAP is enabled on a PPP interface.

```
router# debug serial interface

Attempt to reject authentication ignored.
Serial 0: Unable to respond to CHAP challenge. No USERNAME entry for xxxx
Serial 0: Unable to respond to CHAP challenge. No password defined for \
USERNAME xxx
Serial 0: Failed CHAP authentication with remote.
Serial 0: remote passed CHAP authentication.
Serial 0: Passed CHAP authentication with remote.
```

S2703

Figure 1-74 Sample Debug Serial Interface Output When CHAP Is Enabled on a PPP Interface

The messages listed in Figure 1-74 indicate the current state of CHAP negotiation.

Debug Serial Interface for SMDS Encapsulation

When encapsulation is set to SMDS, **debug serial interface** displays SMDS packets that have been sent and received, as well as any error messages resulting from SMDS packet transmission.

The error messages that the **debug serial interface** command can generate for SMDS follow.

The following message indicates that a new protocol requested SMDS to encapsulate the data for transmission. SMDS does not know yet how to encapsulate the protocol.

```
SMDS: Error on Serial 0, encapsulation bad protocol = x
```

The following message indicates that SMDS was asked to encapsulate a packet, but no corresponding destination E.164 SMDS address was found in any of the static SMDS tables or in the ARP tables:

```
SMDS send: Error in encapsulation, no hardware address, type = x
```

The following message indicates that a protocol such as CLNS or IP has been enabled on an SMDS interface, but the corresponding multicast addresses have not been configured. The *n* variable displays the link type for which encapsulation was requested. This value is only significant to Cisco as an internal protocol type value.

```
SMDS: Send, Error in encapsulation, type=n
```

The following messages can occur when a packet that was somehow corrupted is received on an SMDS interface. The router expected x, but received y.

```
SMDS: Invalid packet, Reserved NOT ZERO, x y
SMDS: Invalid packet, TAG mismatch x y
SMDS: Invalid packet, Bad TRAILER length x y
```

The following messages can indicate an invalid length for an SMDS packet:

```
SMDS: Invalid packet, Bad BA length x
SMDS: Invalid packet, Bad header extension length x
SMDS: Invalid packet, Bad header extension type x
SMDS: Invalid packet, Bad header extension value x
```

The following messages are displayed when the **debug serial interface** command is enabled:

```
Interface Serial 0 Sending SMDS L3 packet:
SMDS: dgsize:x type:0xn src:y dst:z
```

If the **debug serial interface** command is enabled, the following message can be displayed when a packet is received on an SMDS interface, but the destination SMDS address does not match any on that interface:

```
SMDS: Packet n, not addressed to us
```

debug serial packet

Use the **debug serial packet** EXEC command to display more detailed serial interface debugging information than you can obtain using **debug serial interface** command. The **no** form of this command disables debugging output.

debug serial packet
no debug serial packet

Syntax Description

This command has no arguments or keywords.

Command Mode

EXEC

Usage Guidelines

The **debug serial packet** command generates output that is dependent on the type of serial interface and the encapsulation that is running on that interface. The hardware platform also can impact **debug serial packet** output.

Sample Displays

Currently, the **debug serial packet** command displays output for only DDR, PPP, and SMDS encapsulations.

Debug Serial Packet for DDR

When you enable the **debug serial packet** command and 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 Appendix B, “Ethernet Type Codes,” of the *Router Products Command Reference* publication.

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

Debug Serial Packet for PPP

Figure 1-75 shows sample **debug serial packet** output when PPP is enabled on the interface.

```
ppp: config ACK received, type = nnn
ppp: config ACK received, type = nnn, value = yyy
ppp: config ACK received, type = nnn, value = yyy
ppp: config ACK received, type = nnn
ppp: config ACK received, type = nnn
```

S2704

Figure 1-75 Sample Debug Serial Packet Output for PPP

The preceding five messages may appear when PPP is attempting to negotiate a link. They indicate that PPP received an ACK for option type, and if the option has a value, the value that was acked also is displayed. This is possibly useful in debugging PPP link establishment, but is mostly useful with some knowledge of the PPP protocol.

Table 1-55 describes significant fields shown in Figure 1-75.

Table 1-55 Debug Serial Packet Field Descriptions for PPP

Field	Description
ppp: config ACK received	The router has received an acknowledgment packet in response to the configuration negotiation request packet it sent.
type = *nnn*	Number indicating the LCP configuration option to be negotiated. Possible values include: 1—Maximum Received Unit (MRU) 2—Async Control Character Map 3—Authentication Protocol 4—Quality Protocol 5—Magic Number 6—Undefined 7—Protocol Field Compression 8—Address and Control Field Compression 9—32 Bit FCS
value = *yyy*	Value of the LCP configuration option that has been negotiated.

Additional messages that the **debug serial packet** command can generate when PPP is enabled follow.

The following message is displayed when PPP sends a packet onto the line. The “Serial0” shows the interface the packet is sent on. The state corresponds to a PPP state machine state, and is only useful to technical support staff. The link can be either ppp-lcp or ppp-ipcp, indicating that it is either a PPP LCP packet or a PPP IPCP packet. The code is the PPP packet type being transmitted, the ID is a sequence number for this packet, and LEN is the length of packet. This message is only displayed for PPP-generated packets, not for all packets using PPP encapsulation.

```
PPP send: on Serial0 STATE= 4 LINK= ppp-lcp, CODE= 5, ID= 345, LEN = 9
```

The following message is displayed if the PPP timer expires. It indicates that the remote side did not respond to the packet in the time allowed.

```
ppp: TIMEout: Time= 3245532 State= 4
```

The following three messages may be displayed if PPP receives a packet that is incorrectly formatted:

```
ppp: rcvd short header for ppp-lcp
ppp: rcvd illegal length for ppp-lcp
ppp: rcvd short packet. len x > y for ppp-lcp
```

The following message is displayed when a PPP-specific packet is received. It either will be for ppp-lcp or ppp-ipcp, depending on which PPP layer the packet is for. It will give a state, which is a state in the PPP state machine; a code, which is the type of PPP packet received; the ID, which is a sequence number; and the length of the packet.

```
PPP input(ppp-lcp): state = 4 code = 5 id = 345 len = 9
```

The following message is displayed if PPP has received an ACK for a configuration request it transmitted. The ID can be matched with an ID displayed in the PPP send debug message to verify which packet was acked.

```
ppp: state = 4 fsm_rconfack(ppp-lcp): rcvd id 345
```

One of the following messages is displayed when PPP receives a configuration packet from the other side. It displays the configuration type and whether there is a value for that type, the value, and whether it is going to ack, nack, or reject this configuration option.

```
ppp: received config for type = x  value = y acked
ppp: received config for type = x  value = y rejected
ppp: received config for type = x  value = y nacked
```

Debug Serial Packet for SMDS Encapsulation

Figure 1-76 shows sample output when SMDS is enabled on the interface.

```
router# debug serial packet

Interface Serial2 Sending SMDS L3 packet:
SMDS Header  : Id: 00 RSVD: 00 BEtag: EC Basize: 0044
Dest:E18009999999FFFF Src:C12015804721FFFF Xh:040300000300010000000000000000000
SMDS LLC   : AA AA 03 00 00 00 80 38
SMDS Data  : E1 19 01 00 00 80 00 00 0C 00 38 1F 00 0A 00 80 00 00 0C 01 2B 71
SMDS Data  : 06 01 01 0F 1E 24 00 EC 00 44 00 02 00 00 83 6C 7D 00 00 00 00 00
SMDS Trailer  : RSVD: 00 BEtag: EC Length: 0044
```

S2705

Figure 1-76 Sample Debug Serial Packet Output for SMDS

As Figure 1-76 shows, when encapsulation is set to SMDS, **debug serial packet** displays the entire SMDS header (in hex), as well as some payload data on transmit or receive. This information is useful only when you have an understanding of the SMDS protocol. The first line of the output indicates either Sending or Receiving.

debug source-bridge

Use the **debug source-bridge** EXEC command to display information about packets and frames transferred across a source route bridge. The **no** form of this command disables debugging output.

debug source-bridge
no debug source-bridge

Syntax Description

This command has no arguments or keywords.

Command Mode

EXEC

Sample Display

Figure 1-77 shows sample **debug source-bridge** output for peer bridges using TCP as a transport mechanism. The RSRB network configuration has ring 2 and ring 1 bridged together through remote peer bridges. The remote peer bridges are connected via a serial line and use TCP as the transport mechanism.

```
router# debug source-bridge

RSRB: remote explorer to 5/131.108.250.1/1996 srn 2 [C840.0021.0050.0000]
RSRB: Version/Ring XReq sent to peer 5/131.108.250.1/1996
RSRB: Received version reply from 5/131.108.250.1/1996 (version 2)
RSRB: DATA: 5/131.108.250.1/1996 Ring Xchg Rep, trn 2, vrn 5, off 18, len 10
RSRB: added bridge 1, ring 1 for 5/131.108.240.1/1996
RSRB: DATA: 5/131.108.250.1/1996 Explorer trn 2, vrn 5, off 18, len 69
RSRB: DATA: 5/131.108.250.1/1996 Forward trn 2, vrn 5, off 0, len 92
RSRB: DATA: forward Forward srn 2, br 1, vrn 5 to peer 5/131.108.250.1/1996
```

S2706

Figure 1-77 Sample Debug Source Bridge Output in TCP Environment

The following line of output indicates that a remote explorer frame has been sent to IP address 131.108.250.1 and like all RSRB TCP connections, has been assigned port 1996. The bridge belongs to ring group 5. The explorer frame originated from ring number 2. The routing information field (RIF) descriptor has been generated by the local station and indicates that the frame was sent out via bridge 1 onto virtual ring 5.

```
RSRB: remote explorer to 5/131.108.250.1/1996 srn 2 [C840.0021.0050.0000]
```

The following line of output indicates that a request for remote peer information has been sent to IP address 131.108.250.1, TCP port 1996. The bridge belongs to ring group 5.

```
RSRB: Version/Ring XReq sent to peer 5/131.108.250.1/1996
```

The following line of output is the response to the version request previously sent. The response is sent from IP address 131.108.250.1, TCP port 1996. The bridge belongs to ring group 5.

```
RSRB: Received version reply from 5/131.108.250.1/1996 (version 2)
```

The following line of output is the response to the ring request previously sent. The response is sent from IP address 131.108.250.1, TCP port 1996. The target ring number is 2, virtual ring number is 5, the offset is 18, and the length of the frame is 10 bytes.

```
RSRB: DATA: 5/131.108.250.1/1996 Ring Xchg Rep, trn 2, vrn 5, off 0, len 10
```

The following line of output indicates that bridge 1 and ring 1 have been added to the source-bridge table for IP address 131.108.250.1, TCP port 1996.

```
RSRB: added bridge 1, ring 1 for 5/131.108.250.1/1996
```

The following line of output indicates that a packet containing an explorer frame has come across virtual ring 5 from IP address 131.108.250.1, TCP port 1996. The packet is 69 bytes in length. This packet is received after the Ring Exchange information was received and updated on both sides.

```
RSRB: DATA: 5/131.108.250.1/1996 Explorer trn 2, vrn 5, off 18, len 69
```

The following line of output indicates that a packet containing data has come across virtual ring 5 from IP address 131.108.250.1 over TCP port 1996. The packet is being placed on the local target ring 2.The packet is 92 bytes in length.

```
RSRB: DATA: 5/131.108.250.1/1996 Forward trn 2, vrn 5, off 0, len 92
```

The following line of output indicates that a packet containing data is being forwarded to the peer that has IP 131.108.250.1 address belonging to local ring 2 and bridge 1. The packet is forwarded via virtual ring 5. This packet is sent after the Ring Exchange information was received and updated on both sides.

```
RSRB: DATA: forward Forward srn 2, br 1, vrn 5 to peer 5/131.108.250.1/1996
```

Figure 1-78 shows sample **debug source-bridge** output for peer bridges using direct encapsulation as a transport mechanism. The RSRB network configuration has ring 1 and ring 2 bridged together through peer bridges. The peer bridges are connected via a serial line and use TCP as the transport mechanism.

```
router# debug source-bridge

RSRB: remote explorer to 5/Serial1 srn 1 [C840.0011.0050.0000]
RSRB: Version/Ring XReq sent to peer 5/Serial1
RSRB: Received version reply from 5/Serial1 (version 2)
RSRB: IFin: 5/Serial1 Ring Xchg, Rep trn 0, vrn 5, off 0, len 10
RSRB: added bridge 1, ring 1 for 5/Serial1
```

S2707

Figure 1-78 Sample Debug Source Bridge Output in Direct Encapsulation Environment

The following line of output indicates that a remote explorer frame has been sent to remote peer Serial1, which belongs to ring group 5. The explorer frame originated from ring number 1. The routing information field (RIF) descriptor 0011.0050 has been generated by the local station and indicates that the frame was sent out via bridge 1 onto virtual ring 5.

```
RSRB: remote explorer to 5/Serial1 srn 1 [C840.0011.0050.0000]
```

The following line of output indicates that a request for remote peer information has been sent to Serial1. The bridge belongs to ring group 5.

```
RSRB: Version/Ring XReq sent to peer 5/Serial1
```

The following line of output is the response to the version request previously sent. The response is sent from Serial 1. The bridge belongs to ring group 5 and the version is 2.

```
RSRB: Received version reply from 5/Serial1 (version 2)
```

The following line of output is the response to the ring request previously sent. The response is sent from Serial1. The target ring number is 2, virtual ring number is 5, the offset is 0, and the length of the frame is 39 bytes.

```
RSRB: IFin: 5/Serial1 Ring Xchg Rep, trn 2, vrn 5, off 0, len 39
```

The following line of output indicates that bridge 1 and ring 1 have been added to the source-bridge table for Serial1.

```
RSRB: added bridge 1, ring 1 for 5/Serial1
```

debug source event

Use the **debug source event** EXEC command to display information on source-route bridging activity. The **no** form of this command disables debugging output.

debug source event
no debug source event

Syntax Description

This command has no arguments or keywords.

Command Mode

EXEC

Usage Guidelines

Output of the **debug source bridge** command is identical to the output of this command.

Note In order to use the **debug source event** command to display traffic source-routed through an interface, you first must disable fast switching of SRB frames with the **no source-bridge route-cache** interface subcommand.

Sample Display

Figure 1-79 shows sample **debug source event** output.

```
router# debug source event

RSRB0: forward (srn 5 bn 1 trn 10), src: 8110.2222.33c1 dst: 1000.5a59.04f9
[0800.3201.00A1.0050]
RSRB0: forward (srn 5 bn 1 trn 10), src: 8110.2222.33c1 dst: 1000.5a59.04f9
[0800.3201.00A1.0050]
RSRB0: forward (srn 5 bn 1 trn 10), src: 8110.2222.33c1 dst: 1000.5a59.04f9
[0800.3201.00A1.0050]
RSRB0: forward (srn 5 bn 1 trn 10), src: 8110.2222.33c1 dst: 1000.5a59.04f9
[0800.3201.00A1.0050]
RSRB0: forward (srn 5 bn 1 trn 10), src: 8110.2222.33c1 dst: 1000.5a59.04f9
[0800.3201.00A1.0050]
```

S2708

Figure 1-79 Sample Debug Source Event Output

Table 1-56 describes significant fields shown in Figure 1-79.

Table 1-56 Debug Source Event Field Descriptions

Field	Description
RSRB0:	Indicates that this RIF cache entry is for the Token Ring 0 interface, which has been configured for remote source-route bridging. (SRB1, in contrast, would indicate that this RIF cache entry is for Token Ring 1, configured for source-route bridging.)
forward	Indicates that this is a forward (normal data) packet, in contrast to a control packet containing proprietary Cisco bridging information.
srn 5	Indicates the ring number of the packet's source ring.
bn 1	Indicates the bridge number of the bridge this packet traverses.
trn 10	Indicates the ring number of the packet's target ring.
src: 8110.2222.33c1	Source address of the route in this RIF cache entry.
dst: 1000.5a59.04f9	Destination address of the route in this RIF cache entry.
[0800.3201.00A1.0050]	RIF string in this RIF cache entry.

Examples of other **debug source event** messages that can be displayed follow.

In the following example messages, SRB*n* or RSRB*n* denotes a message associated with interface Token Ring *n*. An *n* of 99 denotes the remote side of the network.

```
SRBn: no path, s: <src MAC addr>d: <dst MAC addr>rif: <rif>
```

In the preceding example, a bridgeable packet came in on interface Token Ring *n* but there was nowhere to send it. This is most likely a configuration error. For example, an interface has source bridging turned on, but it is not connected to another source bridging interface or a ring group.

In the following example, a bridgeable packet has been forwarded from Token Ring *n* to the target ring. The two interfaces are directly linked.

```
SRBn: direct forward (srn <ring>bn <bridge>trn <ring>)
```

In the following examples, a proxy explorer reply was not generated because there was no way to get to the address from this interface. The packet came from the node with the first <address>.

```
SRBn: br dropped proxy XID,  <address> for <address>, wrong vring (rem)
SRBn: br dropped proxy TEST, <address> for <address>, wrong vring (rem)
SRBn: br dropped proxy XID,  <address> for <address>, wrong vring (local)
SRBn: br dropped proxy TEST, <address> for <address>, wrong vring (local)
SRBn: br dropped proxy XID,  <address> for <address>, no path
SRBn: br dropped proxy TEST, <address> for <address>, no path
```

In the following example, an appropriate proxy explorer reply was generated on behalf of the second <address>. It is sent to the first <address>.

```
SRBn: br sent proxy XID,  <address> for <address>[<rif>]
SRBn: br sent proxy TEST, <address> for <address>[<rif>]
```

The following example indicates that the broadcast bits were not set, or that the routing information indicator on the packet was not set:

```
SRB<unit#>: illegal explorer, s: <srcMACaddr> d: <destMACaddr> rif:
<RIFstring>
```

The following example indicates that the direction bit in the RIF field was set, or that an odd packet length was encountered. Such packets are dropped.

```
SRB<unit #>: bad explorer control, D set or odd
```

The following example indicates that a spanning explorer was dropped because the spanning option was not configured on the interface:

```
SRB<unit #>: span dropped, input off, s: <src mac addr> d: <dest mac addr>
rif: <rif string>
```

The following example indicates that a spanning explorer was dropped because it had traversed the ring previously:

```
SRB<unit #>: span violation, s: <src mac addr> d: <dest mac addr> rif:
<rif string>
```

The following example indicates that an explorer was dropped because the maximum hop count limit was reached on that interface:

```
SRB<unit #>: max hops reached - <hop cnt>, s: <src mac addr> d: <dest mac addr>
rif: <rif string>
```

The following example indicates that the ring exchange request was sent to the indicated peer. This request tells the remote side which rings this node has and asks for a reply indicating which rings that side has.

```
RSRB: sent RingXreq to <ring group>/<ip addr>
```

The following example indicates that a message has been sent to the remote peer. The <label> variable can be AHDR (active header), PHDR (passive header), HDR (normal header), or DATA (data exchange), and <op> can be Forward, Explorer, Ring Xchg, Req, Ring Xchg, Rep, Unknown Ring Group, Unknown Peer, or Unknown Target Ring.

```
RSRB: <label>: sent <op> to <ring group>/<ip addr>
```

The following example indicates that the remote bridge and ring pair have been removed from or added to the local ring group table because the remote peer has changed:

```
RSRB: removing bn <bridge> rn <ring> from <ring group>/<ip addr>
RSRB: added bridge <bridge>, ring <ring> for <ring group>/<ip addr>
```

The following example shows miscellaneous remote peer connection establishment messages:

```
RSRB: peer <ring group>/<ip addr> closed [last state n]
RSRB: passive open <ip addr>(remote port) -> <local port>
RSRB: CONN: opening peer <ring group>/<ip addr>, attempt n
RSRB: CONN: Remote closed <ring group>/<ip addr> on open
RSRB: CONN: peer <ring group>/<ip addr> open failed, <reason>[code]
```

The following example shows that an explorer packet was propagated onto the local ring from the remote ring group:

```
RSRBn: sent local explorer, bridge <bridge> trn <ring>, [rif]
```

The following messages indicate that the remote source-route bridging code found the packet to be in error:

```
RSRBn: ring group <ring group> not found
RSRBn: explorer rif [rif] not long enough
```

The following example indicates that a buffer could not be obtained for a ring exchange packet; this is an internal error.

```
RSRB: couldn't get pak for ringXchg
```

The following example indicates that a ring exchange packet was received that had an incorrect length; this is an internal error.

```
RSRB: XCHG: req/reply badly formed, length <pak length>, peer <peer id>
```

The following example indicates that a ring entry was removed for the peer; the ring was possibly disconnected from the network, causing the remote router to send an update to all its peers.

```
RSRB: removing bridge <br #> ring <ring #> from <peer name> <ring type>
```

The following example indicates that a ring entry was added for the specified peer; the ring was possibly added to the network, causing the other router to send an update to all its peers.

```
RSRB: added bridge <br #>, ring <ring #> for <peer id>
```

The following example indicates that no memory was available to add a ring number to the ring group specified; this is an internal error.

```
RSRB: no memory for ring element <ring group #>
```

The following example indicates that memory was corrupted for a connection block; this is an internal error.

```
RSRB: CONN: corrupt connection block
```

The following example indicates that a connector process started, but that there was no packet to process; this is an internal error.

```
RSRB: CONN: warning, no initial packet, peer: <ip addr> <peer pointer>
```

The following example indicates that a packet was received with a version number different from the one present on the router:

```
RSRB: IF New version. local=<local version #>, remote=<remote version>,
<pak op code> <peer id>
```

The following example indicates that a packet with a bad op code was received for a direct encapsulation peer; this is an internal error.

```
RSRB: IFin: bad op <op code> (op code string) from <peer id>
```

The following example indicates that the virtual ring header will not fit on the packet to be sent to the peer; this is an internal error:

```
RSRB: vrif_sender, hdr won't fit
```

The following example indicates that the specified peer is being opened. The retry count specifies the number of times the opening operation is attempted.

```
RSRB: CONN: opening peer <peer id> <retry count>
```

The following example indicates that the router, configured for FST encapsulation, received a version reply to the version request packet it had sent previously:

```
RSRB: FST Rcvd version reply from <peer id> (version #)
```

The following example indicates that the router, configured for FST encapsulation, sent a version request packet to the specified peer:

```
RSRB: FST Version Request. op = <opcode>, <peer id>
```

The following example indicates that the router received a packet with a bad op code from the specified peer; this is an internal error.

```
RSRB: FSTin: bad op <opcode> (op code string) from <peer id>
```

The following example indicates that the TCP connection between the router and the specified peer is being aborted:

```
RSRB: aborting <ring group #>/<peer id> (vrtcpd_abort called)
```

The following example indicates that an attempt to establish a TCP connection to a remote peer timed out:

```
RSRB: CONN: attempt timed out
```

The following example indicates that a packet was dropped because the ring group number in the packet did not correlate with the ring groups configured on the router:

```
RSRB<unit #>: ring group <ring group #> not found
```

debug span

Use the **debug span** EXEC command to display information on changes in the spanning-tree topology when debugging a transparent bridge. The **no** form of this command disables debugging output.

debug span
no debug span

Syntax Description

This command has no arguments or keywords.

Command Mode

EXEC

Usage Guidelines

This command is useful for tracking and verifying that the spanning-tree protocol is operating correctly.

Sample Display—IEEE Spanning Tree

Sample **debug span** output for an IEEE BPDU packet follows:

```
ST: Ether4 0000000000000A080002A02D6700000000000A080002A02D67800100001400020 00F00
```

Figure 1-80 shows the preceding **debug span** output broken up by fields and labeled to aid documentation.

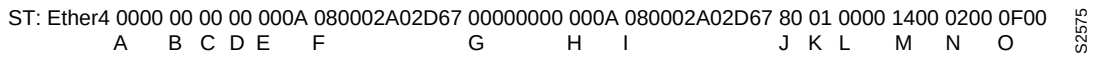


Figure 1-80 Sample Debug Span Output

Table 1-57 describes significant fields shown in this **debug span** output.

Table 1-57 Debug Span Field Descriptions for an IEEE BPDU Packet

Field	Description
ST:	Indicates that this is a spanning tree packet
Ether4	Interface receiving the packet
(A) 0000	Indicates that this is an IEEE BPDU packet
(B) 00	Version
(C) 00	Command Mode 00 indicates config BPDU 80 indicates the Topology Change Notification (TCN) BPDU
(D) 00	Topology change acknowledgment 00 indicates no change 80 indicates a change notification
(E) 000A	Root priority
(F) 080002A02D67	Root ID
(G) 00000000	Root path cost (0 means the sender of this BPDU packet is the root bridge)
(H) 000A	Bridge priority
(I) 080002A02D67	Bridge ID
(J) 80	Port priority
(K) 01	Port No. 1
(L) 0000	Message age in 256ths of a second (0 seconds, in this case)
(M) 1400	Maximum age in 256ths of a second (20 seconds, in this case)
(N) 0200	Hello time in 256ths of a second (2 seconds, in this case)
(O) 0F00	Forward delay in 256ths of a second (15 seconds, in this case)

Sample Display—DEC Spanning Tree

Sample **debug span** output for a DEC BPDU packet follows:

```
ST: Ethernet4 E11901000002000000C01A2C900640080000000C0106CE0A01050F1E6A
```

Figure 1-81 shows the preceding **debug span** output broken up by fields and labeled to aid documentation.

```
E1  19  01  00  0002  00000C01A2C9  0064  0080  00000C0106CE  0A  01  05  0F  1E  6A
A   B   C   D   E     F             G     H     I             J   K   L   M   N   O
                                                                                      S2576
```

Figure 1-81 Sample Debug Span Output

Table 1-58 describes significant fields shown in this **debug span** output.

Table 1-58 Debug Span Field Descriptions for a DEC BPDU Packet

Field	Description
ST:	Indicates that this is a spanning tree packet.
Ethernet4	Interface receiving the packet.
(A) E1	Indicates that this is a DEC BPDU packet.
(B) 19	Indicates that this is a DEC Hello packet. Possible values are as follows: 0x19—DEC Hello 0x02—Topology change notification (TCN)
(C) 01	DEC version.
(D) 00	Flag that is a bit field with the following mapping: 1—TCN 2—TCN acknowledgment 8—Use short timers
(E) 0002	Root priority.
(F) 00000C01A2C9	Root ID (MAC address).
(G) 0064	Root path cost (translated as 100 in decimal notation).
(H) 0080	Bridge priority.
(I) 00000C0106CE	Bridge ID.
(J) 0A	Port ID (in contrast to interface number).
(K) 01	Message age (in seconds).
(L) 05	Hello time (in seconds).
(M) 0F	Maximum age (in seconds).
(N) 1E	Forward delay (in seconds).
(O) 6A	Not applicable.

debug stun packet

Use the **debug stun packet** EXEC command to display information on packets traveling through the STUN links. Use the **no** form of this command to disable debugging output.

debug stun packet [*group*] [*address*]
no debug stun packet [*group*] [*address*]

Syntax Description

group	(Optional.) Decimal integer assigned to a group. Using this option limits output to packets associated with the specified STUN group.
address	(Optional.) Output is further limited to only those packets containing the specified STUN address. The *address* argument is in the appropriate format for the STUN protocol running for the specified group.

Command Mode

EXEC

Usage Guidelines

Because using this command is processor intensive, it is best to use it after hours, rather than in a production environment. It is also best to turn this command on by itself, rather than use it in conjunction with other debug commands.

Sample Display

Figure 1-82 shows sample **debug stun packet** output.

```
router# debug stun packet
STUN sdlc: 0:00:04 Serial3        NDI: (0C2/008) U: SNRM    PF:1
STUN sdlc: 0:00:04 Serial3        NDI: (0C2/008) U: SNRM    PF:1
STUN sdlc: 0:00:01 Serial3        SDI: (0C2/008) U: UA      PF:1
STUN sdlc: 0:00:00 Serial3        SDI: (0C2/008) S: RR      PF:1  NR:000
STUN sdlc: 0:00:00 Serial3        SDI: (0C2/008) S: RR      PF:1  NR:000
STUN sdlc: 0:00:00 Serial3        SDI: (0C2/008) S: RR      PF:1  NR:000
STUN sdlc: 0:00:00 Serial3        SDI: (0C2/008) S: RR      PF:1  NR:000
STUN sdlc: 0:00:00 Serial3        SDI: (0C2/008) S: RR      PF:1  NR:000
STUN sdlc: 0:00:00 Serial3        SDI: (0C2/008) S: RR      PF:1  NR:000
STUN sdlc: 0:00:00 Serial3        SDI: (0C2/008) S: RR      PF:1  NR:000
STUN sdlc: 0:00:00 Serial3        SDI: (0C2/008) S: RR      PF:1  NR:000
STUN sdlc: 0:00:00 Serial3        SDI: (0C2/008) S: RR      PF:1  NR:000
STUN sdlc: 0:00:00 Serial3        SDI: (0C2/008) S: RR      PF:1  NR:000
STUN sdlc: 0:00:00 Serial3        SDI: (0C2/008) S: RR      PF:1  NR:000
STUN sdlc: 0:00:00 Serial3        NDI: (0C2/008) I:         PF:1  NR:000  NS:000
STUN sdlc: 0:00:00 Serial3        SDI: (0C2/008) I:         PF:1  NR:001  NS:000
STUN sdlc: 0:00:00 Serial3        SDI: (0C2/008) S: RR      PF:1  NR:001
STUN sdlc: 0:00:00 Serial3        SDI: (0C2/008) S: RR      PF:1  NR:001
STUN sdlc: 0:00:00 Serial3        SDI: (0C2/008) S: RR      PF:1  NR:001
STUN sdlc: 0:00:00 Serial3        SDI: (0C2/008) S: RR      PF:1  NR:001
```

X1 type of packet (first line of output)
X2 type of packet (fourth line of output)
X3 type of packet (NDI I-frame line)

S2563

Figure 1-82 Sample Debug STUN Packet Output

Explanations for individual lines of output from Figure 1-82 follow.

The following line of output describes an X1 type of packet.

```
STUN sdlc: 0:00:04 Serial3         NDI: (0C2/008) U: SNRM    PF:1
```

Table 1-59 describes significant fields shown in this line of **debug stun packet** output.

Table 1-59 Debug STUN Packet Field Descriptions

Field	Description
STUN sdlc:	Indicates that the STUN feature is providing the information.
0:00:04	Time elapsed since receipt of previous packet.
Serial3	Interface type and unit number reporting the event.
NDI:	Indicates the type of cloud separating the SDLC end nodes. Possible values follow: NDI—Network input SDI—Serial link
0C2	SDLC address of the SDLC connection.
008	Indicates a modulo value of 8.
U:SNRM	Indicates the frame type followed by the command or response type. In this case it is an Unnumbered frame that contains an SNRM (Set Normal Response Mode) command. The possible frame types are as follows: I—Information frame S—Supervisory frame. The possible commands and responses are: RR (Receive Ready), RNR (Receive Not Ready), and REJ (Reject). U—Unnumbered frame. The possible commands are: UI (Unnumbered Information), SNRM, DISC/RD (Disconnect/Request Disconnect), SIM/RIM, XID Exchange Identification), TEST. The possible responses are UA (unnumbered acknowledgment), DM (Disconnected Mode), and FRMR (Frame Reject Mode)
PF:1	Poll/Final bit. 0—Off 1—On

The following line of output describes an X2 type of packet:

```
STUN sdlc: 0:00:00 Serial3         SDI: (0C2/008) S: RR      PF:1  NR:000
```

All of the fields in the previous line of output match those for an X1 type of packet, except the last field, which is additional. NR:000 indicates a receive count of 0; the range for the receive count is 0 to 7.

The following line of output describes an X3 type of packet:

```
STUN sdlc: 0:00:00 Serial3         SDI: (0C2/008) S:I PF:1  NR:000 NS:000
```

All of the fields in the previous line of output match those for an X2 type of packet, except the last field, which is additional. NS:000 indicates a send count of 0; the range for the send count is 0 to 7.

debug tftp

Use the **debug tftp** EXEC command to display TFTP debugging information when encountering problems netbooting or using the **configure network** or **write network** commands. The **no** form of this command disables debugging output.

debug tftp
no debug tftp

Syntax Description

This command has no arguments or keywords.

Command Mode

EXEC

Sample Display

Figure 1-83 shows sample **debug tftp** output from the EXEC command **write network**.

```
router# debug tftp

TFTP: msclock 0x292B4; Sending  write request (retry 0), socket_id 0x301DA8
TFTP: msclock 0x2A63C; Sending  write request (retry 1), socket_id 0x301DA8
TFTP: msclock 0x2A6DC; Received ACK for block 0,   socket_id 0x301DA8
TFTP: msclock 0x2A6DC; Received ACK for block 0,   socket_id 0x301DA8
TFTP: msclock 0x2A6DC; Sending  block 1 (retry 0), socket_id 0x301DA8
TFTP: msclock 0x2A6E4; Received ACK for block 1,   socket_id 0x301DA8
                                                                      S2709
```

Figure 1-83 Sample Debug TFTP Output

Table 1-60 describes significant fields shown in the first line of output from Figure 1-83.

Table 1-60 Debug TFTP Field Descriptions

Message	Description
TFTP:	Indicates that this entry describes a TFTP packet.
msclock 0x292B4;	Internal timekeeping clock (in milliseconds).
Sending write request (retry 0)	Indicates the TFTP operation.
socket_id 0x301DA8	Unique memory address for the socket for the TFTP connection.

debug token ring

Use the **debug token ring** EXEC command to display messages about Token Ring interface activity. The **no** form of this command disables debugging output.

debug token ring
no debug token ring

Syntax Description

This command has no arguments or keywords.

Command Mode

EXEC

Usage Guidelines

This command reports several lines of information for each packet sent or received and is intended for low traffic, detailed debugging.

The Token Ring interface records provide information regarding the current state of the ring. These messages are only displayed when the **debug token events** command is enabled.

The **debug token ring** command invokes verbose Token Ring hardware debugging. This includes detailed displays as traffic arrives and departs the unit.

Note It is best to use this command only on router/bridges with light loads.

Sample Display

Figure 1-84 shows sample **debug token ring** output.

```
router# debug token ring

TR0: Interface is alive, phys. addr 5000.1234.5678
TR0:  in: MAC: acfc: 0x1105 Dst: c000.ffff.ffff Src: 5000.1234.5678  bf: 0x45
TR0:  in:      riflen 0, rd_offset 0, llc_offset 40
TR0: out: MAC: acfc: 0x0040 Dst: 5000.1234.5678 Src: 5000.1234.5678  bf: 0x00
TR0: out: LLC: AAAA0300 00009000 00000100 AAC00000 00000802 50001234 ln: 28
TR0:  in: MAC: acfc: 0x1140 Dst: 5000.1234.5678 Src: 5000.1234.5678  bf: 0x09
TR0:  in: LLC: AAAA0300 00009000 00000100 AAC0B24A 4B4A6768 74732072 ln: 28
TR0:  in:      riflen 0, rd_offset 0, llc_offset 14
TR0: out: MAC: acfc: 0x0040 Dst: 5000.1234.5678 Src: 5000.1234.5678  bf: 0x00
TR0: out: LLC: AAAA0300 00009000 00000100 D1D00000 FE11E636 96884006 ln: 28
TR0:  in: MAC: acfc: 0x1140 Dst: 5000.1234.5678 Src: 5000.1234.5678  bf: 0x09
TR0:  in: LLC: AAAA0300 00009000 00000100 D1D0774C 4DC2078B 3D000160 ln: 28
TR0:  in:      riflen 0, rd_offset 0, llc_offset 14
TR0: out: MAC: acfc: 0x0040 Dst: 5000.1234.5678 Src: 5000.1234.5678  bf: 0x00
TR0: out: LLC: AAAA0300 00009000 00000100 F8E00000 FE11E636 96884006 ln: 28
```

S2710

Figure 1-84 Sample Debug Token Ring Output

Descriptions of sample lines of output in Figure 1-84 follow.

Table 1-61 describes significant fields shown in the second line of output from Figure 1-84.

```
TR0:  in: MAC: acfc: 0x1105 Dst: c000.ffff.ffff Src: 5000.1234.5678  bf: 0x45
```

Table 1-61 Debug Token Ring Field Descriptions—Part 1

Message	Description
TR0:	Name of the interface associated with the Token Ring event.
in:	Indicates whether the packet was input to the interface (in) or output from the interface (out).
MAC:	Indicates the type of packet, as follows: MAC—Media Access Control LLC—Link Level Control
acfc: 0x1105	Access Control, Frame Control bytes, as defined by the IEEE 802.5 standard.
Dst: c000.ffff.ffff	Destination address of the frame.
Src: 5000.1234.5678	Source address of the frame.
bf: 0x45	Bridge flags for internal use by technical support staff.

Table 1-62 describes significant fields shown in the third line of output from Figure 1-84.

```
TR0:  in:      riflen 0, rd_offset 0, llc_offset 40
```

Table 1-62 Debug Token Ring Field Descriptions—Part 2

Message	Description
TR0:	Name of the interface associated with the Token Ring event.
in:	Indicates whether the packet was input to the interface (in) or output from the interface (out).
riflen 0	Length of the RIF field (in bytes).
rd_offset 0	Offset (in bytes) of the frame pointing to the start of the RIF field.
llc_offset 40	Offset in the frame pointing to the start of the LLC field.

Table 1-63 describes significant fields shown in the fifth line of output from Figure 1-84.

```
TR0: out: LLC: AAAA0300 00009000 00000100 AAC00000 00000802 50001234 ln: 28
```

Table 1-63 Debug Token Ring Field Descriptions—Part 3

Message	Description
TR0:	Name of the interface associated with the Token Ring event.
out:	Indicates whether the packet was input to the interface (in) or output from the interface (out).
LLC:	Indicates the type of frame, as follows: MAC—Media Access Control LLC—Link Level Control
AAAA0300....	This and the octets that follow it indicate the contents (hex) of the frame.
ln: 28	Indicates the length of the information field (in bytes).

debug vines arp

Use the **debug vines arp** EXEC command to display debugging information on all ARP packets that the router sends or receives. The **no** form of this command disables debugging output.

debug vines arp
no debug vines arp

Syntax Description

This command has no arguments or keywords.

Command Mode

EXEC

Sample Display

Figure 1-85 shows sample **debug vines arp** output.

```
router# debug vines arp

VNSARP: received ARP type 0 from 0260.8c43.a7e4
VNSARP: sending ARP type 1 to 0260.8c43.a7e4
VNSARP: received ARP type 2 from 0260.8c43.a7e4
VNSARP: sending ARP type 3 to 0260.8c43.a7e4 assigning address 3001153C:8004
VSARP: received ARP type 0 from 0260.8342.1501
VSARP: sending ARP type 1 to 0260.8342.1501
VSARP: received ARP type 2 from 0260.8342.1501
VSARP: sending ARP type 3 to 0260.8342.1501 assigning address 3001153C:8005,
       sequence 143C, metric 2
```

S2711

Figure 1-85 Sample Debug VINES ARP Output

In Figure 1-85, the first line shows that the router received an ARP request (type 0) from station address 0260.8c43.a7e4. The second line shows that the router is sending back the ARP service response (type 1) indicating that it is willing to assign VINES Internet addresses. The third line shows that the router received a VINES Internet address assignment request (type 2) from address 0260.8c43.a7e4. The fourth line shows that the router is responding (type 3) to the address assignment request from the client and assigning it the address 3001153C:8004.

Table 1-64 describes significant fields shown in Figure 1-85.

Table 1-64 Debug VINES ARP Field Descriptions

Field	Description
VINES:	Indicates that this is one of the Banyan VINES debugging messages.
received ARP type 0	Indicates that an ARP request of type 0 was received. Possible type values follow: 0—Query request. The ARP client broadcasts a type 0 message to request an ARP service to respond. 1—Service response. The ARP service responds with a type 1 message to an ARP client's query request. 2—Assignment request. The ARP client responds to a service response with a type 2 message to request a Banyan VINES Internet address. 3—Assignment response. The ARP service responds to an assignment request with a type 3 message that includes the assigned Banyan VINES Internet address.
from 0260.8c43.a7e4	Indicates the source address of the packet.

debug vines echo

Use the **debug vines echo** EXEC command to display information on all MAC-level echo packets that the router sends or receives. Banyan VINES interface testing programs make use of these echo packets. The **no** form of this command disables debugging output.

debug vines echo
no debug vines echo

Syntax Description

This command has no arguments or keywords.

Note These echo packets do not include network layer addresses.

Command Mode

EXEC

Sample Display

Figure 1-86 shows sample **debug vines echo** output.

```
router# debug vines echo

VINESECHO: 100 byte packet from 0260.8c43.a7e4
```

S2712

Figure 1-86 Sample Debug VINES Echo Output

Table 1-65 describes the fields shown in Figure 1-86.

Table 1-65 Debug VINES Echo Field Descriptions

Field	Description
VINESECHO	Indicates that this is a **debug vines echo** message.
100 byte packet	Packet size in bytes.
from 0260.8c43.a7e4	Source address of the echo packet.

debug vines ipc

Use the **debug vines ipc** EXEC command to display information on all transactions that occur at the VINES IPC layer, which is one of the two VINES transport layers. The **no** form of this command disables debugging output.

debug vines ipc
no debug vines ipc

Syntax Description

This command has no arguments or keywords.

Command Mode

EXEC

Usage Guidelines

You can use the **debug vines ipc** command to discover why an IPC layer process on the router is not communicating with another IPC layer process on another router or Banyan VINES server.

Sample Display

Figure 1-87 shows sample **debug vines ipc** output for three pairs of transactions. For more information about these fields or their values, refer to Banyan VINES documentation.

```
router# debug vines ipc

VIPC: sending IPC Data to Townsaver port 7 from port 7
 r_cid 0, l_cid 1, seq 1, ack 0, length 12
VIPC: received IPC Data from Townsaver port 7 to port 7
 r_cid 51, l_cid 1, seq 1, ack 1, length 32
VIPC: sending IPC Ack to Townsaver port 0 from port 0
 r_cid 51, l_cid 1, seq 1, ack 1, length 0
```

S2713

Figure 1-87 Sample Debug VINES IPC Output

Table 1-66 describes the fields shown in Figure 1-87. For more information about these fields or their values, refer to Banyan VINES documentation.

Table 1-66 VINES IPC Field Descriptions

Field	Description
VIPC:	Indicates that this is output from the **debug vines ipc** command.
sending	Indicates that the router is either sending an IPC packet to another router or has received an IPC packet from another router.
IPC Data to	Indicates the type of IPC frame: Acknowledgment Data Datagram Disconnect Error Probe
Townsaver port 7	Indicates the machine name as assigned using the VINES **host** command, or IP address of the other router. Also indicates the port on that machine through which the packet has been transmitted.
from port 7	Indicates the port on the router through which the packet has been transmitted.
r_cid 0, l_cid 1, seq 1, ack 0, length 12	Indicates the values for various fields in the IPC layer header of this packet. Refer to Banyan VINES documentation for more information.

debug vines netrpc

Use the **debug vines netrpc** EXEC command to display information on all transactions that occur at the VINES NetRPC layer, which is the VINES Session/Presentation layer. The **no** form of this command disables debugging output.

debug vines netrpc
no debug vines netrpc

Syntax Description

This command has no arguments or keywords.

Command Mode

EXEC

Usage Guidelines

You can use the **debug vines netrpc** command to discover why a NetRPC layer process on the router is not communicating with another NetRPC layer process on another router or Banyan server.

Sample Display

Figure 1-88 shows sample **debug vines netrpc** output. For more information about these fields or their values, refer to Banyan VINES documentation.

```
router# debug vines netrpc

VRPC: sending RPC call to Townsaver
VRPC: received RPC return from Townsaver
```

S2714

Figure 1-88 Sample Debug VINES NetRPC Output

Table 1-67 describes the fields shown in the first line of output in Figure 1-88. For more information about these fields or their values, refer to Banyan VINES documentation.

Table 1-67 Debug VINES NetRPC Field Descriptions

Field	Description
VRPC:	Indicates that this is output from the **debug vines netrpc** command.
sending RPC	Indicates that the router is either sending a NetRPC packet to another router or has received a NetRPC packet from another router.
call	Indicates the transaction type: abort call reject return return address search search all
Townsaver	Indicates the machine name as assigned using the VINES **host** command or IP address of the other router.

debug vines packet

Use the **debug vines packet** EXEC command to display general VINES debugging information. This information includes packets received, generated, and forwarded, as well as failed access checks and other operations. The **no** form of this command disables debugging output.

debug vines packet
no debug vines packet

Syntax Description

This command has no arguments or keywords.

Command Mode

EXEC

Sample Display

Figure 1-89 shows sample **debug vines packet** output.

```
router# debug vines packet

VINES: s=30028CF9:1 (Ether2), d=FFFFFFFF:FFFF, rcvd w/ hops 0
VINES: s=3000CBD4:1 (Ether1), d=3002ABEA:1 (Ether2), g=3002ABEA:1, sent
VINES: s=3000CBD4:1 (Ether1), d=3000B959:1, rcvd by gw
VINES: s=3000B959:1 (local), d=3000CBD4:1 (Ether1), g=3000CBD4:1, sent
```

S2715

Figure 1-89 Sample Debug VINES Packet Output

The following information describes selected lines of output from Figure 1-89.

Table 1-68 describes the fields shown in the first line of output.

Table 1-68 Debug VINES Packet Field Descriptions

Field	Description
VINES:	Indicates that this is a Banyan VINES packet.
s = 30028CF9:1	Source address of the packet.
(Ether2)	Indicates the interface through which the packet was received.
d = FFFFFFFF:FFFF	Indicates that the destination is a broadcast address.
rcvd w/ hops 0	Indicates that the packet was received because it was a local broadcast packet. The remaining hop count in the packet was zero (0).

In the second line of output that follows, the destination is the address 3002ABEA:1 associated with interface Ether2. Source address 3000CBD4:1 sent a packet to this destination through the gateway at address 3000ABEA:1.

```
VINES: s=3000CBD4:1 (Ether1), d=3002ABEA:1 (Ethernet2), g=3002ABEA:1, sent
```

In the third line of output that follows, the router being debugged is the destination address (3000B959:1).

```
VINES: s=3000CBD4:1 (Ether1), d=3000B959:1, rcvd by gw
```

In the following fifth line of output, (local) indicates that the router being debugged generated the packet.

```
VINES: s=3000B959:1 (local), d=3000CBD4:1 (Ether1), g=3000CBD4:1, sent
```

debug vines routing

Use the **debug vines routing** EXEC command to display information on all RTP update messages sent or received and all routing table activities that occur in the router. The **no** form of this command disables debugging output.

debug vines routing
no debug vines routing

Syntax Description

This command has no arguments or keywords.

Command Mode

EXEC

Sample Display

Figure 1-90 shows sample **debug vines routing** output.

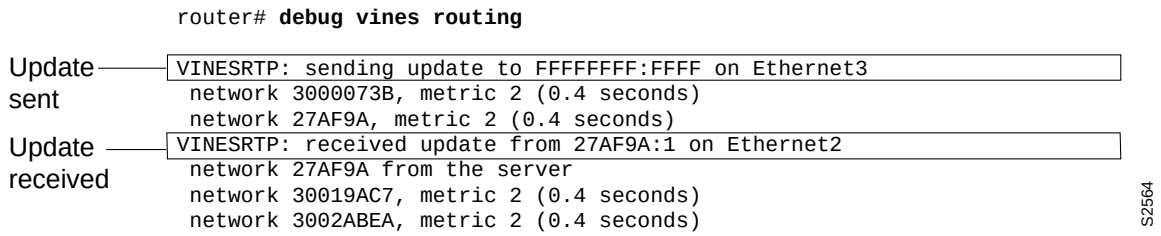


Figure 1-90 Sample Debug VINES Routing Output

Figure 1-90 describes two VINES routing updates; the first includes two entries and the second includes three entries. The following describes selected lines of output from Figure 1-90.

The following first line shows that the router sent a periodic routing update to the broadcast address FFFFFFFF:FFFF through the Ethernet3 interface.

```
VINESRTP: sending update to FFFFFFFF:FFFF on Ethernet3
```

The following second line indicates that the router knows how to reach network 3000073B, which is a metric of 2 away from the router. The value that follows the metric (0.4 seconds) interprets the metric in seconds.

```
network 3000073B, metric 2 (0.4 seconds)
```

The following third line indicates that the router knows how to reach network 27AF9A, which is a metric of 2 away from the router. The value that follows the metric (0.4 seconds) interprets the metric in seconds.

```
network 27AF9A, metric 2 (0.4 seconds)
```

The following fourth line of output indicates that the router received a routing update from the VINES server at VINES address 27AF9A:1 through the Ethernet2 interface.

```
VINESRTP: received update from 27AF9A:1 on Ethernet2
```

The following fifth line of output implies that the server sending this update is directly accessible to the router (even though VINES servers do not explicitly list themselves in routing updates). Because this is an implicit entry in the table, there is no metric associated with this line of output.

```
network 27AF9A from the server
```

As the first actual entry in the routing update from the VINES server at 27AF9A:1, the following sixth line indicates that network 30019AC7 can be reached by sending to this server. This network is a metric of 2 away from the sending server. The value that follows the metric (0.4 seconds) interprets the metric in seconds.

```
network 30019AC7, metric 2 (0.4 seconds)
```

debug vines service

Use the **debug vines service** EXEC command to display information on all transactions that occur at the VINES Service (or applications) layer. The **no** form of this command disables debugging output.

debug vines service
no debug vines service

Syntax Description

This command has no arguments or keywords.

Command Mode

EXEC

Usage Guidelines

You can use the **debug vines service** command to discover why a VINES Service layer process on the router is not communicating with another Service layer process on another router or Banyan server.

Note Because the **debug vines service** command provides the highest level overview of VINES traffic through the router, it is best to begin debugging using this command, and then proceed to use lower-level VINES **debug** commands as necessary.

Sample Display

Figure 1-91 shows sample **debug vines service** output.

```
router# debug vines service

VSRV: Get Time Info sent to Townsaver
VSRV: Get Time Info response from Townsaver, time: 01:47:54 PDT Apr 29 1993
VSRV:     epoch SS@Aloe@Servers-10, age: 0:15:15
```

Sent/ Response pair

S2565

Figure 1-91 Sample Debug VINES Service Output

As Figure 1-91 suggests, **debug vines service** lines of output appear as activity pairs—either a sent/response pair as shown, or as a received/sent pair.

Table 1-69 describes the fields shown in the second line of output in Figure 1-91. For more information about these fields or their values, refer to Banyan VINES documentation.

Table 1-69 Debug VINES Service Field Descriptions—Part 1

Field	Description
VSRV:	Indicates that this is output from the **debug vines service** command.
Get Time Info	Indicates one of three packet types: Get Time Info Time Set Time Sync
response from	Indicates whether the packet was sent to another router, a response from another router, or received from another router.
Townsaver	Indicates the machine name as assigned using the VINES **host** command, or IP address of the other router.
time: 01:47:54 PDT Apr 29 1993	Indicates the current time in hours:minutes:seconds and current date.

Table 1-70 describes the fields shown in the third line of output in Figure 1-91. This line is an extension of the first two lines of output. For more information about these fields or their values, refer to Banyan VINES documentation.

Table 1-70 Debug VINES Service Field Descriptions—Part 2

Field	Description
VSRV:	Indicates that this is output from the **debug vines service** command.
epoch	Indicates that this line of output describes a VINES epoch.
SS@Aloe@Servers-10	Epoch name.
age: 0:15:15	Epoch—elapsed time since the time was last set in the network.

debug vines table

Use the **debug vines table** EXEC command to display information on all modifications to the VINES routing table. The **no** form of this command disables debugging output.

debug vines table
no debug vines table

Syntax Description

This command has no arguments or keywords.

Command Mode

EXEC

Usage Guidelines

This command provides a subset of the information provided by the **debug vines routing** command, as well as some more detailed information on table additions and deletions.

Sample Display

Figure 1-92 shows sample **debug vines table** output.

```
router# debug vines table

VINESRTP: create neighbor 3001153C:8004, interface Ethernet0
```

S2716

Figure 1-92 Sample Debug VINES Table Output

Table 1-71 describes significant fields shown in Figure 1-92.

Table 1-71 Debug VINES Table Field Descriptions

Field	Description
VINESRTP:	Indicates that this is a **debug vines routing** or **debug vines table** message.
create neighbor 3001153C:8004	Indicates that the client at address 3001153C:8004 has been added to the Banyan VINES neighbor table.
interface Ethernet 0	Indicates that this neighbor can be reached through the router interface named Ethernet0.

debug xns packet

Use the **debug xns packet** EXEC command to display information on XNS packet traffic, including the addresses for source, destination, and next hop router of each packet. The **no** form of this command disables debugging output.

debug xns packet
no debug xns packet

Syntax Description

This command has no arguments or keywords.

Command Mode

EXEC

Usage Guidelines

To gain the fullest understanding of XNS routing activity, you should enable **debug xns routing** and **debug xns packet** together.

Sample Display

Figure 1-93 shows sample **debug xns packet** output.

```
router# debug xns packet

XNS: src=5.0000.0c02.6d04, dst=5.ffff.ffff.ffff, packet sent
XNS: src=1.0000.0c00.440f, dst=1.ffff.ffff.ffff, rcvd. on Ethernet0
XNS: src=1.0000.0c00.440f, dst=1.ffff.ffff.ffff, local processing
```

S2717

Figure 1-93 Sample Debug XNS Packet Output.

Table 1-72 describes significant fields shown in Figure 1-93.

Table 1-72 Debug XNS Packet Field Descriptions

Field	Description
XNS:	Indicates that this is an XNS packet.
src = 5.0000.0c02.6d04	Indicates that the source address for this message is 0000.0c02.6d04 on network 5.
dst = 5.ffff.ffff.ffff	Indicates that the destination address for this message is the broadcast address ffff.ffff.ffff on network 5.
packet sent	Indicates that the packet to destination address 5.ffff.ffff.ffff in Figure 1-93, as displayed using the **debug xns packet** command, was queued on the output interface.
rcvd. on Ethernet0	Indicates that the router just received this packet through the Ethernet0 interface.
local processing	Indicates that the router has examined the packet and determined that it must process it, rather than forwarding it.

debug xns routing

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

debug xns routing
no debug xns routing

Syntax Description

This command has no arguments or keywords.

Command Mode

EXEC

Usage Guidelines

To gain the fullest understanding of XNS routing activity, enable **debug xns routing** and **debug xns packet** together.

Sample Display

Figure 1-94 shows sample **debug xns routing** output.

```
router# debug xns routing

XNSRIP: sending standard periodic update to 5.ffff.ffff.ffff via Ethernet2
  network 1, hop count 1
  network 2, hop count 2

XNSRIP: got standard update from 1.0000.0c00.440f socket 1 via Ethernet0
  net 2: 1 hops
```

S2718

Figure 1-94 Sample Debug XNS Routing Output

Table 1-73 describes significant fields shown in Figure 1-94.

Table 1-73 Debug XNS Routing Field Descriptions

Field	Description
XNSRIP:	Indicates that this is an XNS routing packet.
sending standard periodic update	The router indicates that this is a periodic XNS routing information update.
to 5.ffff.ffff.ffff	Indicates that the destination address is ffff.ffff.ffff on network 5.
via Ethernet2	Name of the output interface.
network 1, hop count 1	Indicates that network 1 is one hop away from this router.
got standard update from 1.0000.0c00.440f	The router indicates that it has received an XNS routing information update from address 0000.0c00.440f on network 1.
socket 1	The socket number is a well-known port for XNS. Possible values include: 1—routing information 2—echo 3—router error

debug x25 all

Use the **debug x25 all** EXEC command to display information on all X.25 traffic, this includes data, control messages, and flow control (RR and RNR) packets. The **no** form of this command disables debugging output.

debug x25 all
no debug x25 all

Syntax Description

This command has no arguments or keywords.

Command Mode

EXEC

Usage Guidelines

This command is particularly useful for diagnosing problems encountered when placing CALLs.

The **debug x25 all** output includes data, control messages and flow control packets for all of the router's virtual circuits. The **debug x25 events** and **debug x25 vc** commands provide a subset of this output.

Caution Because **debug x25 all** displays all X.25 traffic, it is processor intensive and can render the router useless. Only use **debug x25 all** when the aggregate of all X.25 traffic is fewer than five packets per second.

Sample Display

Figure 1-95 shows sample **debug x25 all** output.

```
router# debug x25 all

Serial2 (236414440): X25 O R3 RESTART (5) 8 lci 0 cause 7 diag 0
Serial2 (236414444): X25 I R3 RESTART (5) 8 lci 0 cause 0 diag 0
Serial2 (236424436): X25 I P1 CALL REQUEST (11) 8 lci 1024
 From(2): 49 To(2): 46
 Facilities: (0)
 First byte of call user data (4): 0xCC
Serial2 (236424440): X25 O P4 CALL CONNECTED (3) 8 lci 1024
Serial2 (236426444): X25 I P4 DATA (103) 8 lci 1024 PS 0 PR 0
Serial2 (236426448): X25 O D1 DATA (103) 8 lci 1024 PS 0 PR 1
Serial2 (236426460): X25 I D1 DATA (103) 8 lci 1024 PS 1 PR 0
Serial2 (236426464): X25 O D1 DATA (103) 8 lci 1024 PS 1 PR 2
Serial2 (236426484): X25 I D1 RR (3) 8 lci 1024 PR 2
Serial2 (236426500): X25 I D1 DATA (103) 8 lci 1024 PS 2 PR 2
Serial2 (236426500): X25 O D1 DATA (103) 8 lci 1024 PS 2 PR 3
Serial2 (236453060): X25 I D1 CLEAR REQUEST (5) 8 lci 1024 cause 0 diag 122
Serial2 (236453060): X25 O D1 CLEAR CONFIRMATION (3) 8 lci 1024
X25-Switch (274428): X25 O D1 PVC-SETUP, wait to connect (29) <Serial2 pvc 3><Serial2 pvc 1> 2/1 128/64
X25-Switch (274432): X25 I D1 PVC-SETUP, connected (29) <Serial2 pvc 3><Serial2 pvc 1> 2/1 128/64
Serial2 (236453064): X25 O D1 RESET REQUEST (5) 8 lci 3 cause 15 diag 0
Serial2 (236453064): X25 1 D1 RESET CONFIRMATION (3) 8 lci 3
```

S2719

Figure 1-95 Sample Debug X25 All Output

Figure 1-95 shows a typical exchange of packets between two X.25 devices on a network. The first line of output in Figure 1-95, shown below, describes a RESTART packet.

```
Serial2 (236414440): X25 O R3 RESTART (5) 8 lci 0 cause 7 diag 0
```

Table 1-74 describes the fields in this line of output.

Table 1-74 Debug X25 All Field Descriptions

Field	Description
Serial2	Indicates the interface associated with this X.25 event.
(236414440)	System clock (in milliseconds). Useful for determining the amount of time between events.
X25	Indicates that this message describes an X.25 event.
O	Indicates whether the X.25 message was input (I) or output (O) through the interface.
R3	State of the virtual circuit. Possible values follow. D1—Flow control ready D2—DTE reset request D3—DCE reset indication P1—Idle P2—DTE waiting for DCE to connect CALL P3—DCE waiting for DTE to accept CALL P4—Data transfer P5—CALL collision P6—DTE clear request P7—DCE clear indication R1—Packet level ready R2—DTE restart request R3—DCE restart indication X1—Nonstandard state for a virtual circuit in hold-down See Annex B of the 1984 CCITT X.25 Recommendation for more information on these states.

Field	Description
RESTART	Describes the type of X.25 packet. Possible values follow. CALL CONNECTED CALL REQUEST CLEAR CONFIRMATION CLEAR REQUEST DATA DIAGNOSTIC ILLEGAL INTR CONFIRMATION INTR (interrupt) PVC-SETUP REGISTRATION REGISTRATION CONFIRMATION RESET CONFIRMATION RESET REQUEST RESTART RESTART CONFIRMATION RNR (Receiver Not Ready) RR (Receiver Ready)
(5)	Number of bytes in the packet.
8	Modulo of the virtual circuit. Possible values are 8 or 128.
lci 0	Virtual circuit number. See Annex A of the 1984 CCITT X.25 Recommendation for information on VC assignment.
cause 7	Code indicating the event that triggered the packet. The cause field can only appear in entries for CLEAR REQUEST, RESET REQUEST, and RESTART packets. Possible values for the cause field can vary, depending on the type of packet. Refer to Appendix A of this manual, “X.25 Cause and Diagnostics Codes,” for explanations of these codes.
diag 0	Code providing an additional hint as to what, if anything, went wrong. The diag field can only appear in entries for CLEAR REQUEST, DIAGNOSTIC (as “error 0”), RESET REQUEST and RESTART packets. Because of the large number of possible values, they are listed in Appendix A of this manual, “X.25 Cause and Diagnostic Codes.”

Notice that the first DATA packet in Figure 1-95 contains two fields not yet documented.

```
Serial2 (236426444): X25 I P4 DATA (103) 8 lci 1024 PS 0 PR 0
```

Table 1-75 describes the PS and PR fields that can appear in a **debug x25 all** display.

Table 1-75 Debug X25 PS and PR Field Descriptions

Field	Description
PS 0	Packet send sequence number; used for flow control of the outgoing packet stream. Present only in DATA packets.
PR 0	Packet receive sequence number; used for flow control of the incoming packet. stream. Present only in DATA, RR, and RNR packets.

In Figure 1-95, notice also that the CALL REQUEST packet precedes three other lines of output that have a unique format.

```
Serial2 (236424436): X25 I P1 CALL REQUEST (11) 8 lci 1024
From(2): 49 To(2): 46
Facilities: (0)
First byte of call user data (4): 0xCC
```

These lines indicate that the CALL REQUEST packet has a two-digit source address, 49, and a two-digit destination address, 46. These are X.121 addresses that can be from 0 to 15 digits in length. The Facilities field is (0) bytes in length, indicating that no X.25 facilities are being requested. The optional call user data field is 4 bytes in length. The first of these bytes has the hexadecimal value of CC, indicating that the caller intends for IP datagrams to be carried on the VC.

The two lines of output in Figure 1-95 that begin with X25-Switch are shown below.

```
X25-Switch (274428): X25 O D1 PVC-SETUP, wait to connect (29) <Serial2 pvc 3><Serial2 pvc
1> 2/1 128/64
X25-Switch (274432): X25 I D1 PVC-SETUP, connected (29) <Serial2 pvc 3><Serial2 pvc 1>
2/1 128/64
```

These lines of output do not describe standard X.25 packets. Instead, they describe proprietary Cisco messages that represent a tunneled PVC setup between two routers. Table 1-76 describes the fields these two lines of output.

Table 1-76 Debug X25 All Field Descriptions for Packets Representing Tunneled PVC Activity

Field	Description
X25-Switch	Indicates that this message travels over a TCP connection.
(274428)	System clock (in milliseconds). Useful for determining the amount of time between events.
X25	Indicates that this message describes an X.25 event.
O	Indicates whether the X.25 message was input (I) or output (O) through the interface.
D1	State of the permanent virtual circuit. Possible values follow. D1—Flow control ready D2—DTE reset request D3—DCE reset indication See Annex B of the 1984 CCITT X.25 Recommendation for more information on these states.

Field	Description
waiting to connect	State of the PVC. Some of these strings only apply to PVCs that are remotely tunneled over a TCP connection. The %X25-3-PVCBAD system error message (as documented in the *System Error Messages* publication), and the **show x25 vc** command (as documented in the *Router Products Command Reference* publication) also use these PVC state strings. Possible values follow: awaiting PVC-SETUP reply can't support flow control values connected dest. disconnected dest. interface is not up dest. PVC configuration mismatch mismatched flow control values no such dest. interface no such dest. PVC non-X.25 dest. interface PVC setup protocol error PVC/TCP connect timed out PVC/TCP connection refused PVC/TCP routing error trying to connect via TCP waiting to connect
(29)	Incoming/outgoing message size (in bytes).
<Serial2 pvc 3>	Interface and PVC port that originated the message (originator).
<Serial2 pvc 1>	Interface and PVC port that responded to that message (responder).
2/1	Window size (in packets).
128/64	Maximum packet size (in bytes).

debug x25 events

Use the **debug x25 events** EXEC command to display information on all X.25 traffic except X.25 data or acknowledgment packets. The **no** form of this command disables debugging output.

debug x25 events
no debug x25 events

Syntax Description

This command has no arguments or keywords.

Command Mode

EXEC

Usage Guidelines

The **debug x25 events** command is useful for debugging X.25 problems, because it shows changes that occur in the virtual circuits handled by the router. Because most X.25 connectivity problems stem from errors that CLEAR or RESET virtual circuits, you can use **debug x25 events** to identify these errors.

While **debug x25 all** output includes both data and control messages for all of the router's virtual circuits, **debug x25 events** output includes only control messages for all of the router's VCs. In contrast, **debug x25 vc** output includes only control messages for a particular VC. Thus, **debug x25 events** output is a subset of **debug x25 all** output, and **debug x25 vc** output is a subset of **debug x25 events** output.

Note Because **debug x25 events** displays a subset of all X.25 traffic, it is safer to use than **debug x25 all** during production hours.

Sample Display

Figure 1-96 shows sample **debug x25 events** output.

```
router# debug x25 events

Serial2 (236543528): X25 I R3 RESTART (5) 8 lci 0 cause 0 diag 0
Serial2 (236552660): X25 I P1 CALL REQUEST (11) 8 lci 1024
 From(2): 49 To(2): 46
 Facilities: (0)
 First byte of call user data (4): 0xCC
Serial2 (236552664): X25 O P4 CALL CONNECTED (3) 8 lci 1024
Serial2 (236564056): X25 I D1 CLEAR REQUEST (5) 8 lci 1024 cause 0 diag 122
Serial2 (236564056): X25 O D1 CLEAR CONFIRMATION (3) 8 lci 1024
Serial2 (236564060): X25 I D1 RESET REQUEST (5) 8 lci 1 cause 0 diag 122
Serial2 (236564060): X25 O D1 RESET CONFIRMATION (3) 8 lci 1
```

S2720

Figure 1-96 Sample Debug X25 Events Output

See the **debug x25 all** command description for information on the fields in **debug x25 events** output.

debug x25 vc

Use the **debug x25 vc** EXEC command to display information on traffic for a particular virtual circuit in order to solve any connectivity or performance problems it is exhibiting. The **no** form of this command disables debugging output.

debug x25 vc *number*
no debug x25 vc *number*

Syntax Description

number	LCI number associated with the virtual circuit(s) you want to monitor.

Command Mode

EXEC

Usage Guidelines

Because no interface is specified, traffic on any VC that has the specified *number* is reported.

While **debug x25 all** output includes both data and control messages for all of the router's virtual circuits, **debug x25 events** output includes only control messages for all of the router's VCs. In contrast, **debug x25 vc** output includes only control messages for a particular VC. Thus, **debug x25 events** output is a subset of **debug x25 all** output, and **debug x25 vc** output is a subset of **debug x25 events** output.

Note Because **debug x25 vc** only displays traffic for a small subset of virtual circuits, it is safe to use even under heavy traffic conditions, as long as events for that virtual circuit are fewer than 25 packets per second.

Sample Display

Figure 1-97 shows sample **debug x25 vc** output.

```
router# debug x25 events
X25 special event debugging is on
router# debug x25 vc 1
X25 debugging output restricted to VC1
router# show debug
X.25 (debugging restricted to VC number 1):
 X25 special event debugging is on

Serial0: X25 0 P2 CALL REQUEST (19) 8 lci 1
 From(14): 31250000000101 To(14): 31109090096101
 Facilities (0)
Serial0: X25 I P2 CLEAR REQUEST (5) 8 lci 1 cause diag 122
```

S2721

Figure 1-97 Sample Debug X25-VC Output

See the **debug x25 all** command description for information on the fields in **debug x25 vc** output.