debug ipx ipxwan

Use the **debug ipx ipxwan** EXEC command to display debug information for interfaces configured to use IPXWAN. The **no** form of this command disables debugging output.

debug ipx ipxwan
no debug ipx ipxwan

Syntax Description

This command has no arguments or keywords.

Command Mode

EXEC

Usage Guidelines

The **debug ipx ipxwan** command is useful for verifying the startup negotiations between two routers running the IPX protocol through a WAN. This command produces output only during state changes or startup. During normal operations, no output is produced.

Sample Display

Figure 2-53 shows sample **debug ipx ipxwan** output during link startup.

```
router# debug ipx ipxwan

%LINEPROTO-5-UPDOWN: Line protocol on Interface Serial1, changed state to up
IPXWAN: state (Disconnect -> Sending Timer Requests) [Serial1/6666:200 (IPX line
 state brought up)]
IPXWAN: state (Sending Timer Requests -> Disconnect) [Serial1/6666:200 (IPX line
 state brought down)]
IPXWAN: state (Disconnect -> Sending Timer Requests) [Serial1/6666:200 (IPX line
 state brought up)]

IPXWAN: Send TIMER_REQ [seq 0] out Serial1/6666:200
IPXWAN: Send TIMER_REQ [seq 1] out Serial1/6666:200
IPXWAN: Send TIMER_REQ [seq 2] out Serial1/6666:200
IPXWAN: Send TIMER_REQ [seq 0] out Serial1/6666:200

IPXWAN: Rcv TIMER_REQ on Serial1/6666:200, NodeID 1234, Seq 1
IPXWAN: Send TIMER_REQ [seq 1] out Serial1/6666:200
IPXWAN: Rcv TIMER_RSP on Serial1/6666:200, NodeID 1234, Seq 1, Del 6
IPXWAN: state (Sending Timer Requests -> Master: Sent RIP/SAP) [Serial1/6666:200
 (Received Timer Response as master)]
IPXWAN: Send RIPSAP_INFO_REQ [seq 0] out Serial1/6666:200
IPXWAN: Rcv RIPSAP_INFO_RSP from Serial1/6666:200, NodeID 1234, Seq 0
IPXWAN: state (Master: Sent RIP/SAP -> Master: Connect) [Serial1/6666:200 (Received
Router
Info Rsp as Master)]
```

Figure 2-53 Sample Debug IPX IPXWAN Output

Explanations for representative lines of output in Figure 2-53 follow.

The following line indicates that the interface has initialized:

```
%LINEPROTO-5-UPDOWN: Line protocol on Interface Serial1, changed state to up
```

The following lines indicate that the startup process failed to receive a timer response, brought the link down, then brought the link up and tried again with a new timer set:

```
IPXWAN: state (Sending Timer Requests -> Disconnect) [Serial1/6666:200 (IPX line
 state brought down)]
IPXWAN: state (Disconnect -> Sending Timer Requests) [Serial1/6666:200 (IPX line
 state brought up)]
```

The following lines indicate that the interface is sending timer requests and waiting on timer response:

```
IPXWAN: Send TIMER_REQ [seq 0] out Serial1/6666:200
IPXWAN: Send TIMER_REQ [seq 1] out Serial1/6666:200
```

The following lines indicate that the interface has received a timer request from the other end of the link and has sent a timer response. The fourth line shows that the interface has come up as the master on the link.

```
IPXWAN: Rcv TIMER_REQ on Serial1/6666:200, NodeID 1234, Seq 1
IPXWAN: Send TIMER_REQ [seq 1] out Serial1/6666:200
IPXWAN: Rcv TIMER_RSP on Serial1/6666:200, NodeID 1234, Seq 1, Del 6
IPXWAN: state (Sending Timer Requests -> Master: Sent RIP/SAP) [Serial1/6666:200
 (Received Timer Response as master)]
```

The following lines indicate that the interface is sending RIP/SAP requests:

```
IPXWAN: Send RIPSAP_INFO_REQ [seq 0] out Serial1/6666:200
IPXWAN: Rcv RIPSAP_INFO_RSP from Serial1/6666:200, NodeID 1234, Seq 0
IPXWAN: state (Master: Sent RIP/SAP -> Master: Connect) [Serial1/6666:200 (Received
Router Info Rsp as Master)]
```

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 non-fast switched packets will produce output. When the IPX cache is invalidated or cleared, one packet for each destination is displayed as the cache is repopulated.

Sample Display

Figure 2-54 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
```

Figure 2-54 Sample Debug IPX Packet Output

In Figure 2-54, the first line indicates that the router receives a packet from a 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 2-30 describes significant fields shown in Figure 2-54.

Table 2-30 Debug IPX Packet Field Descriptions

Field	Description
IPX	Indication 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 to display 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 2-55 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
```

Figure 2-55 Sample Debug IPX Routing Output

Table 2-31 describes significant fields shown in Figure 2-55.

Table 2-31 Debug IPX Routing Field Descriptions

Field	Description
IPXRIP	This is an IPX RIP packet.
update from 9999.0260.8c6a.1733	This packet is a routing update from a Novell server at address 9999.0260.8c6a.1733.
110801 in 1 hops	Network 110801 is one hop away from the router at address 9999.0260.8c6a.1733.
delay 2	Delay is 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.
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	The packet includes routing update information for network 555.
metric 2	Network 555 is two metrics (or hops) away from the router.
delay 3	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.

Related Command

debug ipx sap

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 [**activity** | **events**]
no debug ipx sap

Syntax Description

activity	(Optional) Provides more detailed output of SAP packets, including displays of services in SAP packets.
events	(Optional) Limits amount of detailed output for SAP packets to those that contain interesting events.

Command Mode

EXEC

Usage Guidelines

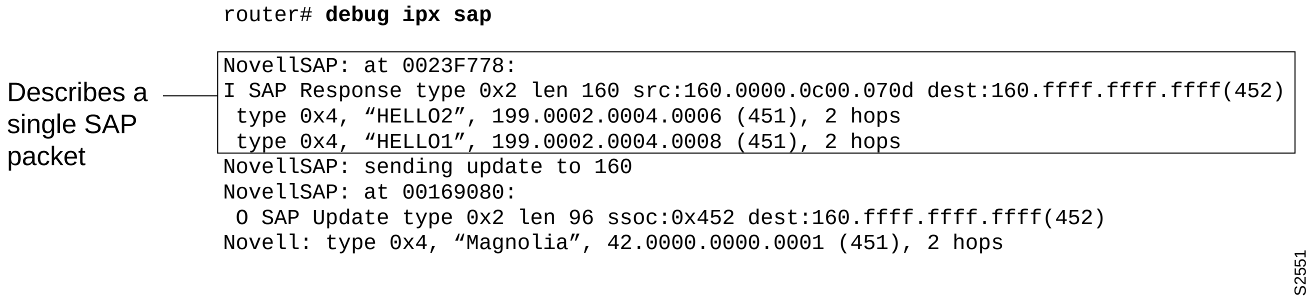
Normally, a router or server sends out one SAP update per minute. Each SAP packet can include up to seven entries. If many servers are advertising on the network, the router sends out multiple packets per update. For example, if a router has 20 entries in the SAP table, it would send three SAP packets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

Obtain the most meaningful detail by using the **debug ipx sap activity** and the **debug ipx sap events** commands together.

Caution Because the **debug ipx sap** command can generate a lot of output, use it with caution on networks that have many interfaces and large service tables.

Sample Display

Figure 2-56 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
```

Figure 2-56 Sample Debug IPX SAP Output

As Figure 2-56 shows, the **debug ipx sap** command generates multiple lines of output for each SAP packet—a packet summary message and a service detail message.

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

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

Table 2-32 describes the fields shown in the second line of output in Figure 2-56.

Table 2-32 Debug IPX SAP Field Descriptions—Part 1

Field	Description
I	Indication as to whether the router received the SAP packet as input (I) or is sending an update as output (O).
SAP Response type 0x2	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 2-33 describes the fields shown in the third and fourth lines of output in Figure 2-56.

Table 2-33 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 21—NAS SNA gateway 24—Remote bridge server 2D—Time Synchronization VAP 2E—Dynamic SAP 47—Advertising print server 4B—Btrieve VAP 5.0 4C—SQL VAP 7A—TES—NetWare for VMS 98—NetWare access server 9A—Named Pipes server 9E—Portable NetWare—UNIX 111—Test server 166—NetWare management 233—NetWare management agent 237—NetExplorer NLM 239—HMI hub 23A—NetWare LANalyzer agent 26A—NMS management FFFF—Wildcard (any SAP service) Contact Novell for more information.
"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 2-56 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 2-34 describes possible values for the ssoc: field.

Table 2-34 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 Integrated Services Digital Network (ISDN) events occurring on the user side (on the router) of the ISDN interface. The ISDN events that can be displayed 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 isdn-event** and the **debug isdn-q931** commands provide similar debug information, the information is displayed in a different format. If you want to see the information in both formats, enable both 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

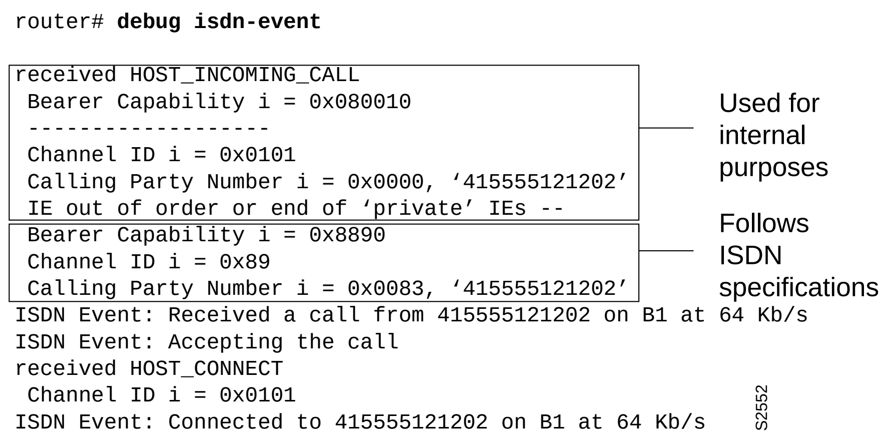
Figure 2-57 shows sample **debug isdn-event** output of call setup events for an outgoing call.

```
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
```

Figure 2-57 Sample Debug ISDN-Event Output—Call Setup Outgoing Call

Figure 2-58 shows sample **debug isdn-event** output of call setup events for an incoming call. The values used for internal purposes are unpacked information elements. The values that follow the ISDN specification are an interpretation of the unpacked information elements. Refer to the "ISDN Switch Types, Codes, and Values" appendix for information about these values.



```
router# debug isdn-event

received HOST_INCOMING_CALL
 Bearer Capability i = 0x080010
 -------------------
 Channel ID i = 0x0101
 Calling Party Number i = 0x0000, '415555121202'
 IE out of order or end of 'private' IEs --
 Bearer Capability i = 0x8890
 Channel ID i = 0x89
 Calling Party Number i = 0x0083, '415555121202'
ISDN Event: Received a call from 415555121202 on B1 at 64 Kb/s
ISDN Event: Accepting the call
received HOST_CONNECT
 Channel ID i = 0x0101
ISDN Event: Connected to 415555121202 on B1 at 64 Kb/s
```

Figure 2-58 Sample Debug ISDN-Event Output—Call Setup Incoming Call

Figure 2-59 shows sample **debug isdn-event** output of call teardown events for a call that has been hung up by the other side of the connection.

```
router# debug isdn-event

received HOST_DISCONNECT
ISDN Event: Call to 415555121202 was hung up
```

Figure 2-59 Sample Debug ISDN-Event Output—Call Teardown by Far End

Figure 2-60 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.

```
router# debug isdn-event

ISDN Event: Hangup call to call id 0x8008
```

Figure 2-60 Sample Debug ISDN-Event Output—Call Teardown Local Side

Table 2-35 describes significant fields shown in Figure 2-57 through Figure 2-60.

Table 2-35 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 ITU-T[1] 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.
Calling Party Number	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 415555121202 on 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.

1. The ITU-T carries out the functions of the former Consultative Committee for International Telegraph and Telephone.

Figure 2-61 shows sample **debug isdn-event** output of a call teardown event for a call that has passed call screening then has been hung up by the ISDN interface on the far end side.

```
0:04:51: 291.848 RX <-  DISCONNECT pd = 8  callref = 0x83
0:04:51:         Cause i = 0x8090 - Normal call clearing
```

Figure 2-61 Sample Debug ISDN-Event—Call Screening Normal Disconnect

Figure 2-62 shows sample **debug isdn-event** output of a call teardown event for a call that has not passed call screening and has been rejected by the ISDN interface on the router side.

```
0:06:44: 404.732 RX <-  DISCONNECT pd = 8  callref = 0x82
0:06:44:         Cause i = 0x8095 - Call rejected
```

Figure 2-62 Sample Debug ISDN-Event—Call Screening Call Rejection

Figure 2-63 shows sample **debug isdn-event** output of a call teardown event for an outgoing call that uses a dialer subaddress.

```
0:04:55: ISDN Event: Call to 5551201:123
0:04:55: 295.692 TX ->  SETUP pd = 8  callref = 0x02
0:04:55:         Bearer Capability i = 0x8890
0:04:55:         Channel ID i = 0x83
0:04:55:         Called Party Number i = 0x80, '5551201'
0:04:55:         Called Party SubAddr i = 0x80, 'P123'
0:04:55: 295.840 RX <-  CALL_PROC pd = 8  callref = 0x82
0:04:55:         Channel ID i = 0x89
0:04:55: received HOST_PROCEEDING
        Channel ID i = 0x0101
0:04:55:        -------------------
        Channel ID i = 0x89
0:04:56: 296.044 RX <-  CONNECT pd = 8  callref = 0x82
0:04:56: received HOST_CONNECT
        Channel ID i = 0x0101
0:04:56:        -------------------
0:04:56: ISDN Event: Connected to 5551201:123 on B1 at 64 Kb/s
0:04:56: 296.064 TX ->  CONNECT_ACK pd = 8  callref = 0x02.
```

Figure 2-63 Sample Debug ISDN-Event Display—Called Party Subaddress

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 ITU-T recommendation Q.921. The **debug isdn-q921** command output 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.

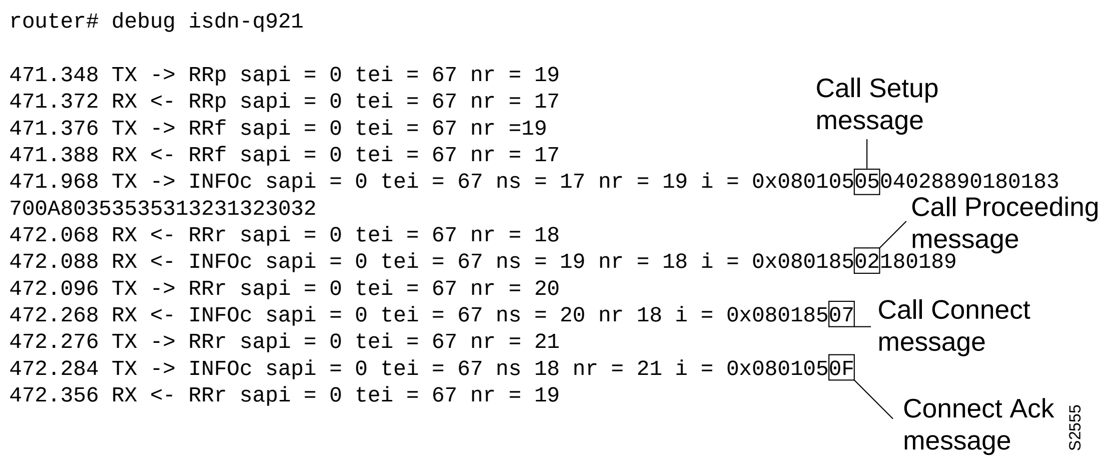
Note The ISDN switch provides the network interface defined by Q.921. This debug command does not display data link layer access procedures taking place within the ISDN network (that is, procedures taking place on the network side of the ISDN connection). See the "ISDN Switch Types, Codes, and Values"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

Sample Display

Figure 2-64 shows sample **debug isdn-q921** output for an outgoing call.

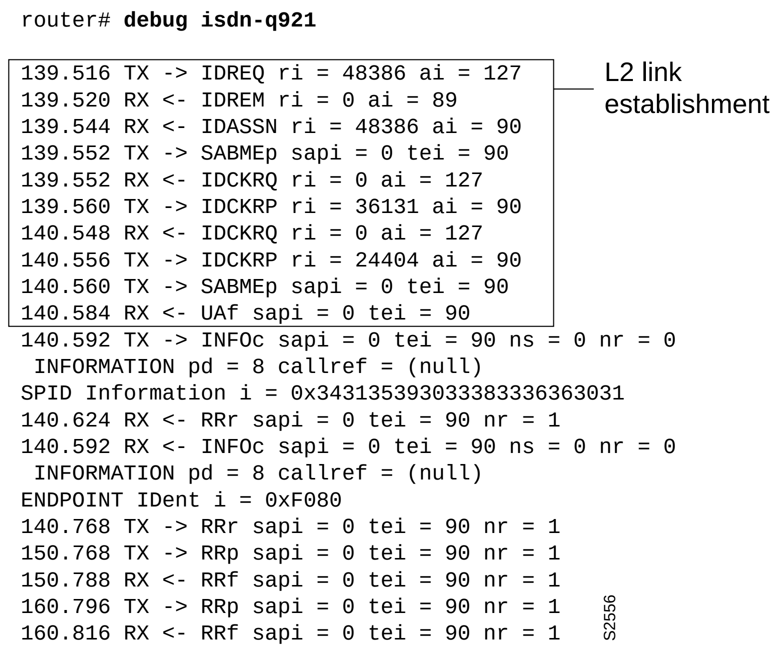


```
router# debug isdn-q921

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 = 0x08010505040288901801832
700A80353535313231323032
472.068 RX <- RRr sapi = 0 tei = 67 nr = 18
472.088 RX <- INFOc sapi = 0 tei = 67 ns = 19 nr = 18 i = 0x08018502180189
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
```

Figure 2-64 Sample Debug ISDN-Q921 Output for Outgoing Call

Figure 2-65 shows sample **debug isdn-q921** output for a startup message on a DMS-100 switch.



```
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 = 0x34313539303338333636303031
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
```

Figure 2-65 Sample Debug ISDN-Q921 Output for Startup Message on a DMS-100 Switch

Figure 2-66 shows sample **debug isdn-q921** output for an incoming call. It is an incoming SETUP message that assumes the L2 link is already established 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 =
0x08010805040288900180018 96C100083383130313233344536373839 3032
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
```

Figure 2-66 Debug ISDN-Q921 Output for Incoming Call

Table 2-36 describes significant fields in Figure 2-64, Figure 2-65, and Figure 2-66.

Table 2-36 Debug ISDN-Q921 Field Descriptions

Field	Description
139.516	Indicates the time, in seconds, 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.
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 Identity Request 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 service access point identifier (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.
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.

Field	Description
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 is 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 ITU-T 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 the “ISDN Switch Types, Codes, and Values” appendix.
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.
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.

Field	Description
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 2-64 follow.

The following lines 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 among user devices that request 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 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 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 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 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 ITU-T recommendation Q.931, supplemented by other specifications such as for switch types VN2 and VN3.The router tracks only activities that occur on the user side, not the network side, of the network connection. The display information **debug isdn-q931** command output 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.

You can use the **debug isdn-q931** command with the **debug isdn-event** and the **debug isdn-q921** commands at the same time. The displays will be intermingled.

Sample Display

Figure 2-67 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)
```

Figure 2-67 Sample Debug ISDN-Q931 Output—Call Setup Procedure for an Outgoing Call

Figure 2-68 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
```

Figure 2-68 Sample Debug ISDN-Q931 Output—Call Setup Procedure for an Incoming Call

Figure 2-69 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
```

Figure 2-69 Sample Debug ISDN-Q931 Output—Call Teardown Procedure from the Network

Figure 2-70 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
```

Figure 2-70 Sample Debug ISDN-Q931 Output—Call Teardown Procedure from the Router

Table 2-37 describes significant fields in Figure 2-67 through Figure 2-70.

Table 2-37 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 to distinguishes messages for call control over the user-network ISDN interface from other ITU-T-defined messages, including other Q.931messages. The protocol discriminator is 8 for call control messages such as SETUP. For basic-1tr6, the protocol discriminator is 65.
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 ITU-T 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 ITU-T Recommendation Q.931.
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 ITU-T recommendation Q.931 for detailed information about DISCONNECT cause codes and RELEASE cause codes.
Looking 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 release 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 2-71 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
ISIS-Adj: Rec L2 IIH from 0000.0c00.0c36 (Ethernet1), cir type 3, cir id
BBBB.BBBB.BBBB.03
```

Figure 2-71 Sample Debug ISIS Adj Packets Output

Explanations for individual lines of output from Figure 2-71 follow.

The following line 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 interprets as 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 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 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 (IS-IS) Intra-Domain Routing Exchange Protocol (IDRP) 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 provides information for determining how long 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 2-72 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
```

Figure 2-72 Sample Debug ISIS SPF Statistics Output

Table 2-38 describes significant fields shown in Figure 2-72.

Table 2-38 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 2-72 follow.

The following lines 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 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 detected by a 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 2-73 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
```

Figure 2-73 Sample Debug ISIS Update-Packets Output

Explanations for individual lines of output from Figure 2-73 follow.

The following lines 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 indicate that the network service access point (NSAP) identified as 8888.8800.0181.00 was 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 indicate that the NSAP identified as 8888.8800.0181.00 was 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 indicates that the router 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 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 indicate that a Level 2 partial sequence number PDU (PSNP) has been received on Tunnel0:

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

The following line indicates that a Level 2 PSNP 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 interfaces 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 2-74 shows sample **debug lapb** output. (The numbers 1 through 7 at the top of the display have been added in order to aid documentation.)

```
   1      2   3    4     5    6    7
Serial0: LAPB I CONNECT (5) IFRAME P 2 1
Serial0: LAPB O REJSENT (2) REJ F 3
Serial0: LAPB O REJSENT (5) IFRAME 0 3
Serial0: LAPB I REJSENT (2) REJ (C) 7
Serial0: LAPB I DISCONNECT (2) SABM P
Serial0: LAPB O CONNECT (2) UA F
Serial0: LAPB O CONNECT (5) IFRAME 0 0
Serial0: LAPB T1 CONNECT 357964 0
```

Figure 2-74 Sample Debug LAPB Output

In Figure 2-74 each line of output describes a LAPB event. There are two types of LAPB events: frame events (when a frame enters or exits the LAPB) and timer events. In Figure 2-74, the last line describes a timer event; all of the other lines describe frame events. Table 2-39 describes the first seven fields shown in Figure 2-74.

Table 2-39 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	Frame event type. Possible values follow: I—Frame input O—Frame output T1—T1 timer expired T3—Interface outage timer expired T4—Idle link 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)

Field	Description
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 SABME—Set asynchronous balanced mode, extended UA—Unnumbered acknowledgment In a T1 timer event, this value is the number of retransmissions already attempted.
Seventh field (Note that this field will not print if the frame control field is required to appear as either a command or a response, and that frame type is correct.)	This field is only present in frame events. It describes the frame type identified by the LAPB address and Poll/Final bit. Possible values are as follows: (C)—Command frame (R)—Response frame P—Command/Poll frame F—Response/Final frame /ERR—Command/Response type is invalid for the control field. An ?ERR generally means that the DTE/DCE assignments are not correct for this link. BAD-ADDR—Address field is neither Command nor Response

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 2-74 follow.

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 (modulo 8 operation) or 0 and 127 (for modulo 128 operation) 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 error information is decoded to display the rejected control field, V(R) and V(S) values, the Response/Command flag, and the error flags WXYZ.

In the following example, the output shows an idle link timer action (T4) where the timer expires twice on an idle link, with the value of T4 set to five seconds:

```
Serial2: LAPB T4 CONNECT 255748
Serial2: LAPB O CONNECT (2) RR P 5
Serial2: LAPB I CONNECT (2) RR F 5
Serial2: LAPB T4 CONNECT 260748
Serial2: LAPB O CONNECT (2) RR P 5
Serial2: LAPB I CONNECT (2) RR F 5
```

The next example shows an interface outage timer expiration (T3):

```
Serial2: LAPB T3 DISCONNECT 273284
```

The following example output shows an error condition when no DCE to DTE connection exists. Note that if a frame has only one valid type (for example, a SABM can only be a command frame), a received frame that has the wrong frame type will be flagged as a receive error (R/ERR in the following output). This feature makes misconfigured links (DTE-DTE or DCE-DCE) easy to spot. Other, less common errors will be highlighed too, such as a too-short or too-long frame, or an invalid address (neither command nor response):

```
Serial2: LAPB T1 SABMSENT 1026508 1
Serial2: LAPB O SABMSENT (2) SABM P
Serial2: LAPB I SABMSENT (2) SABM (R/ERR)
Serial2: LAPB T1 SABMSENT 1029508 2
Serial2: LAPB O SABMSENT (2) SABM P
Serial2: LAPB I SABMSENT (2) SABM (R/ERR)
```

The output in the next example shows the router is misconfigured and has a standard (modulo 8) interface connected to an extended (modulo 128) interface. This condition is indicated by the SABM balanced mode and SABME balanced mode extended messages appearing on the same interface:

```
Serial2: LAPB T1 SABMSENT 1428720 0
Serial2: LAPB O SABMSENT (2) SABME P
Serial2: LAPB I SABMSENT (2) SABM P
Serial2: LAPB T1 SABMSENT 1431720 1
Serial2: LAPB O SABMSENT (2) SABME P
Serial2: LAPB I SABMSENT (2) SABM P
```

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 2-75 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
```

Figure 2-75 Sample Debug LAT Packet Output

The second line of output in Figure 2-75 describes a packet that is input to the router. Table 2-40 describes the fields in this line.

Table 2-40 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 = A	Indicates the message type (in hex). Possible values are as follows:
	0 = Run Circuit
	1 = Start Circuit
	2 = Stop Circuit
	A = Service Announcement
	C = Command
	D = Status
	E = Solicit Information
	F = Response Information

The third line of output in Figure 2-75 describes a packet that is output from the router. Table 2-41 describes the last three fields in this line.

Table 2-41 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 lex rcmd

Use the **debug lex rcmd** EXEC command to debug LAN Extender (LEX) remote commands. The **no** form of this command disables debugging output.

debug lex rcmd
no debug lex rcmd

Syntax Description

This command has no arguments or keywords.

Command Mode

EXEC

Sample Display

Figure 2-76 shows sample **debug lex rcmd** output.

```
router# debug lex rcmd

LEX-RCMD: "shutdown" command received on unbound serial interface- Serial0
LEX-RCMD: Lex0 : "inventory" command received
Rcvd rcmd: FF 03 80 41 41 13 00 1A 8A 00 00 16 01 FF 00 00
Rcvd rcmd: 00 02 00 00 07 5B CD 15 00 00 0C 01 15 26
LEX-RCMD: ACK or response received on Serial0 without a corresponding ID
LEX-RCMD: REJ received
LEX-RCMD: illegal CODE field received in header: <number>
LEX-RCMD: illegal length for Lex0 : "lex input-type-list"
LEX-RCMD: Lex0 is not bound to a serial interface
LEX-RCMD: encapsulation failure
LEX-RCMD: timeout for Lex0: "lex priority-group" command
LEX-RCMD: re-transmitting Lex0: "lex priority-group" command
LEX-RCMD: lex_setup_and_send called with invalid parameter
LEX-RCMD: bind occurred on shutdown LEX interface
LEX-RCMD: Serial0- No free Lex interface found with negotiated MAC address 0000.0c00.d8db
LEX-RCMD: No active Lex interface found for unbind
```

Figure 2-76 Sample Debug LEX Rcmd Output

Explanations for individual lines of output from Figure 2-76 follow.

The following output indicates that a LEX remote command packet was received on a serial interface which is not bound to a LEX interface.

```
LEX-RCMD: "shutdown" command received on unbound serial interface- Serial0
```

This message can occur for any of the LEX remote commands. Possible causes of this message are as follows:

- FLEX state machine software error
- Serial line momentarily goes down, which is detected by the host but not by FLEX

The following output indicates that a LEX remote command response has been received. The hexadecimal values are for internal use only:

```
LEX-RCMD: Lex0 : "inventory" command received
Rcvd rcmd: FF 03 80 41 41 13 00 1A 8A 00 00 16 01 FF 00 00
Rcvd rcmd: 00 02 00 00 07 5B CD 15 00 00 0C 01 15 26
```

The following output indicates that when the host router originates a LEX remote command to FLEX, it generates an 8-bit identifier which is used to associate a command with its corresponding response:

```
LEX-RCMD: ACK or response received on Serial0 without a corresponding ID
```

This message could be displayed for any of the following reasons:

- FLEX was very busy at the time that the command arrived and could not send an immediate response. The command timedout on the host router and then FLEX finally sent the response.
- Transmission error.
- Software error.

Possible responses to Config-Request are Config-ACK, Config-NAK, and Config-Rej. The following output shows that some of the options in the Config-Request are not recognizable or are not acceptable to FLEX due to transmission errors or software errors:

```
LEX-RCMD: REJ received
```

The following output shows that a LEX remote command response was received but that the CODE field in the header was incorrect:

```
LEX-RCMD: illegal CODE field received in header: <number>
```

The following output indicates that a LEX remote command response was received but that it had an incorrect length field. This message can occur for any of the LEX remote commands:

```
LEX-RCMD: illegal length for Lex0 : "lex input-type-list"
```

The following output shows that a host router was about to send a remote command when the serial link went down:

```
LEX-RCMD: Lex0 is not bound to a serial interface
```

The following output shows that the serial interface's encapsulation routine failed to encapsulate the remote command datagram because the LEX-NCP was not in the OPEN state. Due to the way the PPP state machine is implemented, it is normal to see a single encapsulation failure for each remote command that gets sent at bind time.

```
LEX-RCMD: encapsulation failure
```

The following output shows that the timer expired for the given remote command without having received a response from the FLEX device. This message can occur for any of the LEX remote commands:

```
 LEX-RCMD: timeout for Lex0: "lex priority-group" command
```

This message could be displayed for any of the following reasons:

- FLEX too busy to respond
- Transmission failure
- Software error

The following output indicates that the host is retransmitting the remote command after a timeout:

```
LEX-RCMD: re-transmitting Lex0: "lex priority-group" command
```

The following output indicates that an illegal parameter was passed to the lex_setup_and_send routine. This message could be displayed for due to a host software error:

```
LEX-RCMD: lex_setup_and_send called with invalid parameter
```

The following output is informational and shows when a bind occurs on a shutdown interface:

```
LEX-RCMD: bind occurred on shutdown LEX interface
```

The following output shows that LEX-NCP reached the open state and a bind operation was attempted with the FLEX's MAC address, but that no free LEX interfaces were found that were configured with that MAC address. This output can occur when the network administrator does not configure a LEX interface with the correct MAC address.

```
LEX-RCMD: Serial0- No free Lex interface found with negotiated MAC address 0000.0c00.d8db
```

The following output shows that the serial line that was bound to the LEX interface went down, and the unbind routine was called, but when the list of active LEX interfaces was searched the LEX interface corresponding to the serial interface was not found. This output usually occurs because of a host software error:

```
LEX-RCMD: No active Lex interface found for unbind
```

debug lnm events

Use the **debug lnm events** EXEC command to display any unusual events that occur on a Token Ring network. These events include 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 2-77 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
```

Figure 2-77 Sample Debug LNM Events Output

Explanations for the messages shown in Figure 2-77 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 detect whether 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 received a response 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 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 message indicates that the router has attempted to determine whether 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 follow for other messages that the **debug lnm events** command can generate.

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 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 station 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 *n*”.

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 2-78 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
```

Figure 2-78 Sample Debug LNM LLC Output

As Figure 2-78 indicates, **debug lnm llc** output can vary somewhat in format. Table 2-42 describes significant fields shown in the first line of output in Figure 2-78.

Table 2-42 Debug LNM LLC Field Descriptions

Field	Description
IBMNM:	This line of output displays LLC-level debugging information.
Received	The router received a frame. The other possible value is Sending, to indicate that the router is sending a frame.
LRM	The function of the LLC-level software that 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 2-78 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 2-79 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.
```

Figure 2-79 Sample Debug LNM MAC Output

Table 2-43 describes significant fields shown in the first line of output in Figure 2-79.

Table 2-43 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	Indicates the 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	Indicates the source address of the frame, if the router has received the frame. If the router is sending the frame, this address is the destination address of the frame.

As Figure 2-79 indicates, all **debug lnm mac** messages follow the format described in Table 2-43 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 2-80 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
LACK_STATE: 2497644, hashp 2AE628, old state = awaiting LLC2 open to finish,
new state = connected
```

Figure 2-80 Sample Debug Local-Ack State Output

Table 2-44 describes significant fields shown in Figure 2-80.

Table 2-44 Debug Local-Ack State Field Descriptions

Field	Description
LACK_STATE:	Indication 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	The old state condition in the local acknowledgment state machine. Possible values include the following: Disconn (disconnected) awaiting LLC2 open to finish connected awaiting linkdown response
new state = awaiting LLC2 open to finish	The new state condition in the local acknowledgment state machine. Possible values include the following: 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 2-81 illustrates a collection of sample **debug netbios-name-cache** 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
NETBIOS: Cannot duplicate packet in netbios_name_cache_proxy
```

Figure 2-81 Sample Debug NetBIOS-Name-Cache Output

Note The sample display provided in Figure 2-81 is a composite output. Debugging output that you actually see would not necessarily occur in this sequence.

Table 2-45 describes selected **debug netbios-name-cache** output fields.

Table 2-45 Debug NetBIOS-Name-Cache Field Descriptions

Field	Description
NETBIOS	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	Indication 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 2-81.

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 lines indicate that an invalid NetBIOS entry exists 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 line indicates that the router 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 line indicates that the NetBIOS name ORINDA is in the name cache table and was updated to the current value:

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

The following line 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 line 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 line 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 line indicates that the entry for ORINDA was flushed from the cache table:

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

The following line indicates that the entry for ORINDA timed out and was flushed from the cache table:

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

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

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

The following line 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 line indicates that the system could not find a NetBIOS name in the cache:

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

The following line indicates that the system found the destination NetBIOS name in the cache, but located on the same ring from which the packet came. The router will drop this packet because the packet should not leave this ring.

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

The following line indicates that the system found the NetBIOS name 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 line 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 can not 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 2-82 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
```

Figure 2-82 Sample Debug Packet Output

Table 2-46 describes significant fields shown in Figure 2-82.

Table 2-46 Debug Packet Field Descriptions

Field	Description
Ethernet0	Name of the Ethernet interface that received the packet.
Unknown	The network could not classify this packet. Examples include packets with unknown link types.
ARPA	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 and Password Authentication Protocol (PAP) 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
- Causes for PAP session failures

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

Sample Displays

Figure 2-83 shows 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
```

Figure 2-83 Sample Debug PPP Packet Output

Table 2-47 describes significant fields shown in Figure 2-83.

Table 2-47 Debug PPP Packet Field Descriptions

Field	Description
PPP	This is PPP debugging output.
Serial4	Interface number associated with this debugging information.
(o), O	This packet was detected as an output packet.
(i) I	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)	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)	Echo Request; value in parentheses is the hexadecimal representation of the LCP type.
LCP ECHOREP (A)	Echo Reply; value in parentheses is the hexadecimal representation of the LCP type.

To elaborate on the displayed output, consider the partial exchange in Figure 2-84. 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
```

Figure 2-84 Partial Debug PPP Packet 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 2-85 shows 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
```

Figure 2-85 Sample Debug PPP Negotiation Output

Table 2-48 describes significant fields shown in Figure 2-85.

Table 2-48 Debug PPP Negotiation Field Descriptions

Field	Description
ppp	This is a PPP debugging output.
sending CONFREQ	The router sent a configuration request.
type = 4 (CI_QUALITYTYPE)	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	The receiving node has received the proposed option negotiation for the indicated option type.
acked	Acknowledgment and acceptance of options.
state = ACKSENT	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 2-85.

The first two lines in Figure 2-85 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 accept 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 lines indicate that the router received a CONFACK from the responding side and displays accepted option values. Use the rcvd id field to verify that 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
```

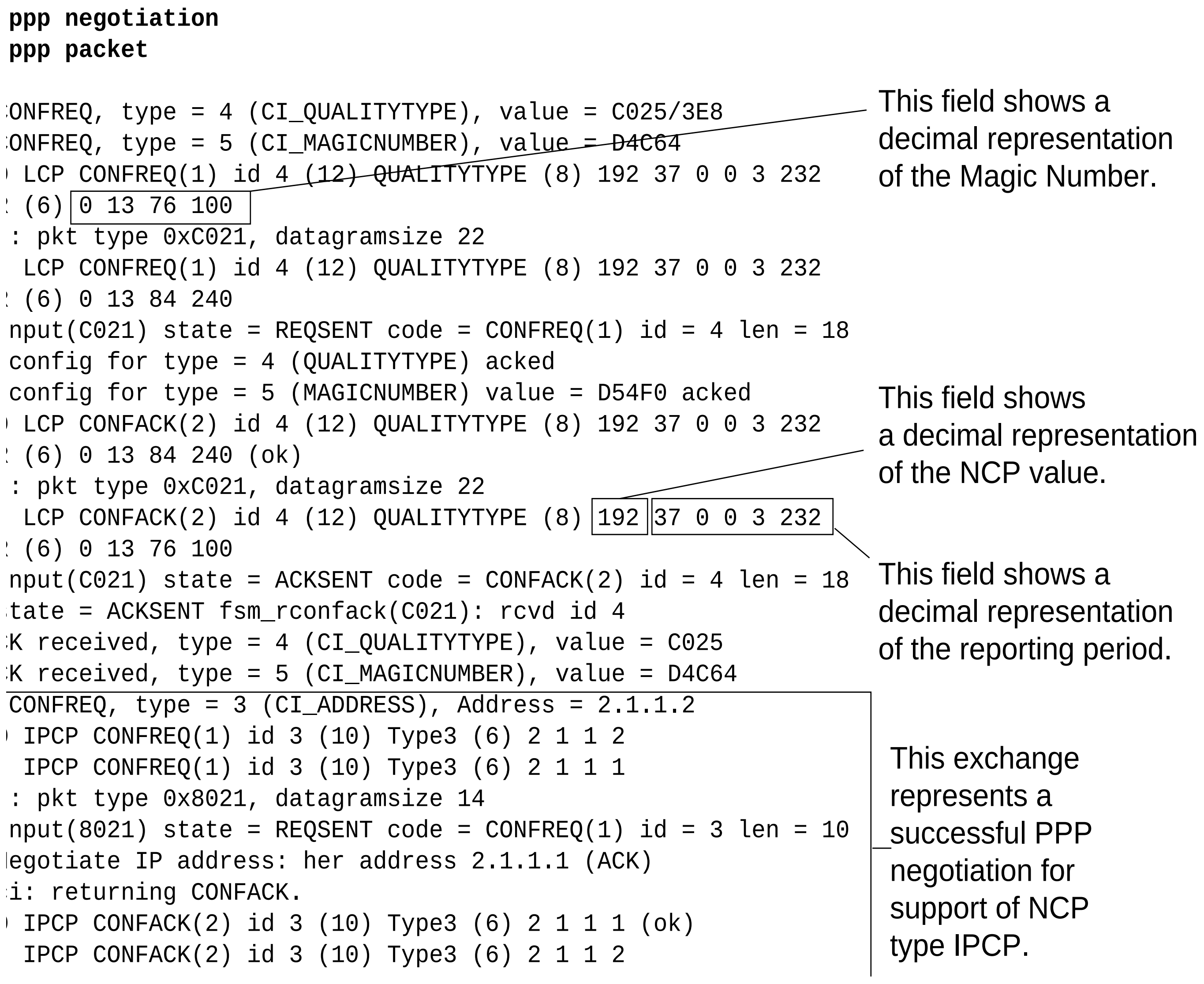
The next line indicates that the router has IP routing enabled on this interface and that the IPCP NCP negotiated successfully:

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

In the last line, the router's state is listed as ACKSENT.

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

Figure 2-86 shows sample output when **debug ppp packet** and **debug ppp negotiation** output are enabled at the same time.



```
ppp negotiation
ppp packet

:ONFREQ, type = 4 (CI_QUALITYTYPE), value = C025/3E8
:ONFREQ, type = 5 (CI_MAGICNUMBER), value = D4C64
' LCP CONFREQ(1) id 4 (12) QUALITYTYPE (8) 192 37 0 0 3 232
: (6) 0 13 76 100
: pkt type 0xC021, datagramsize 22
 LCP CONFREQ(1) id 4 (12) QUALITYTYPE (8) 192 37 0 0 3 232
: (6) 0 13 84 240
nput(C021) state = REQSENT code = CONFREQ(1) id = 4 len = 18
config for type = 4 (QUALITYTYPE) acked
config for type = 5 (MAGICNUMBER) value = D54F0 acked
' LCP CONFACK(2) id 4 (12) QUALITYTYPE (8) 192 37 0 0 3 232
: (6) 0 13 84 240 (ok)
: pkt type 0xC021, datagramsize 22
 LCP CONFACK(2) id 4 (12) QUALITYTYPE (8) 192 37 0 0 3 232
: (6) 0 13 76 100
nput(C021) state = ACKSENT code = CONFACK(2) id = 4 len = 18
tate = ACKSENT fsm_rconfack(C021): rcvd id 4
:K received, type = 4 (CI_QUALITYTYPE), value = C025
:K received, type = 5 (CI_MAGICNUMBER), value = D4C64
CONFREQ, type = 3 (CI_ADDRESS), Address = 2.1.1.2
' IPCP CONFREQ(1) id 3 (10) Type3 (6) 2 1 1 2
 IPCP CONFREQ(1) id 3 (10) Type3 (6) 2 1 1 1
: pkt type 0x8021, datagramsize 14
nput(8021) state = REQSENT code = CONFREQ(1) id = 3 len = 10
legotiate IP address: her address 2.1.1.1 (ACK)
:i: returning CONFACK.
' IPCP CONFACK(2) id 3 (10) Type3 (6) 2 1 1 1 (ok)
 IPCP CONFACK(2) id 3 (10) Type3 (6) 2 1 1 2
```

This field shows a decimal representation of the Magic Number.
This field shows a decimal representation of the NCP value.
This field shows a decimal representation of the reporting period.
This exchange represents a successful PPP negotiation for support of NCP type IPCP.

Figure 2-86 Sample Debug PPP Output with Packet and Negotiation Options Enabled

Figure 2-87 shows sample **debug ppp negotiation** 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
```

Figure 2-87 Sample Debug PPP Negotiation Output When No Response Is Detected

Figure 2-88 shows sample output when no response is detected for configuration requests (with both **debug ppp negotiation** and **debug ppp packet** enabled).

```
router# debug ppp negotiation
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
```

Figure 2-88 Sample Debug PPP Output When No Response Is Detected (with Negotiation and Packet Enabled)

Figure 2-89 shows 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.
```

Figure 2-89 Sample Debug PPP Error Output

Table 2-49 describes significant fields shown in Figure 2-89.

Table 2-49 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 2-90 shows sample **debug ppp chap** output. When doing CHAP authentication, use this **debug** command to determine why an authentication fails. This command is also useful when doing PAP authentication.

```
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
```

Figure 2-90 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 2-50.

Table 2-50 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	The following messages can appear: 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 qllc error

Use the debug qllc error EXEC command to display QLLC errors. The **no** form of this command disables debugging output.

debug qllc error
no debug qllc error

Syntax Description

This command has no arguments or keywords.

Command Mode

EXEC

Usage Guidelines

This command is helpful to track down errors in the QLLC interactions with X.25 networks. Use **debug qllc error** in conjunction with **debug x25** to see the connection. The data shown by this command only flows through the router on the X.25 connection. Some forms of this command can generate lots of output and network traffic.

Sample Display

Figure 2-91 shows sample **debug qllc** output.

```
router# debug qllc error

%QLLC-3-GENERRMSG: qllc_close - bad qllc pointer Caller 00407116 Caller 00400BD2
QLLC 4000.1111.0002: NO X.25 connection.  Dicarding XID and calling out
```

Figure 2-91 Sample Debug QLLC Error Output

Explanations for individual lines of output from Figure 2-91 follow.

The following line indicates that the QLLC connection was closed:

```
%QLLC-3-GENERRMSG: qllc_close - bad qllc pointer Caller 00407116 Caller 00400BD2
```

The following line shows the virtual MAC address of the failed connection:

```
QLLC 4000.1111.0002: NO X.25 connection.  Dicarding XID and calling out
```

debug qllc packet

Use the **debug qllc packet** EXEC command to display QLLC events and QLLC data packets. The **no** form of this command disables debugging output.

debug qllc packet
no debug qllc packet

Syntax Description

This command has no arguments or keywords.

Command Mode

EXEC

Usage Guidelines

This command is helpful to track down errors in the QLLC interactions with X.25 networks. The data shown by this command only flows through the router on the X. Use **debug qllc packet** in conjunction with **debug x25** to see the connection and the data that flows through the router.

Sample Display

Figure 2-92 shows sample **debug qllc packet** output.

```
router# debug qllc packet

14:38:05: Serial2/5 QLLC I: Data Packet.-RSP    9 bytes.
14:38:07: Serial2/6 QLLC I: Data Packet.-RSP  112 bytes.
14:38:07: Serial2/6 QLLC O: Data Packet.  128 bytes.
14:38:08: Serial2/6 QLLC I: Data Packet.-RSP    9 bytes.
14:38:08: Serial2/6 QLLC I: Data Packet.-RSP  112 bytes.
14:38:08: Serial2/6 QLLC O: Data Packet.  128 bytes.
14:38:08: Serial2/6 QLLC I: Data Packet.-RSP    9 bytes.
14:38:12: Serial2/5 QLLC I: Data Packet.-RSP  112 bytes.
14:38:12: Serial2/5 QLLC O: Data Packet.  128 bytes.
```

Figure 2-92 Sample Debug QLLC Packet Output

Explanations for individual lines of output from Figure 2-92 follow.

The following lines indicate a packet was received on the interfaces:

```
14:38:05: Serial2/5 QLLC I: Data Packet.-RSP    9 bytes.
14:38:07: Serial2/6 QLLC I: Data Packet.-RSP  112 bytes.
```

The following lines show that a packet was transmitted on the interfaces:

```
14:38:07: Serial2/6 QLLC O: Data Packet.  128 bytes.
14:38:12: Serial2/5 QLLC O: Data Packet.  128 bytes.
```

debug qllc timer

Use the **debug qllc timer** EXEC command to display QLLC timer events. The **no** form of this command disables debugging output.

debug qllc timer
no debug qllc timer

Syntax Description

This command has no arguments or keywords.

Command Mode

EXEC

Usage Notes

The QLLC process peridocally cycles and checks status of itself and its partner. If the partner is not found in the desired state, a LAPB primitive command is resent until the partner is in the desired state or the timer expires.

Sample Display

Figure 2-93 shows sample **debug qllc timer** output.

```
router# debug qllc timer

14:27:24: Qllc timer lci 257, state ADM retry count 0 Caller 00407116 Caller 00400BD2
14:27:34: Qllc timer lci 257, state NORMAL retry count 0
14:27:44: Qllc timer lci 257, state NORMAL retry count 1
14:27:54: Qllc timer lci 257, state NORMAL retry count 1
```

Figure 2-93 Sample Debug QLLC Timer Output

Explanations for individual lines of output from Figure 2-93 follow.

The following line of output shows the state of a QLLC partner on a given X.25 logical channel identifier:

```
14:27:24: Qllc timer lci 257, state ADM retry count 0 Caller 00407116 Caller 00400BD2
```

Other messages are informational and appear every ten seconds.

debug qllc x25

Use the debug qllc x25 EXEC command to display X.25 packets that affect a QLLC connection. The **no** form of this command disables debugging output.

debug qllc x25
no debug qllc x25

Syntax Description

This command has no arguments or keywords.

Command Mode

EXEC

Usage Guidelines

This command is helpful to track down errors in the QLLC interactions with X.25 networks. Use **debug qllc x25** in conjunction with **debug x25** to see the X.25 events between the router and its partner.

Sample Display

Figure 2-94 shows sample **debug qllc** output.

```
router# debug qllc x25
qllc x.25 events debugging is on

15:07:23: QLLC X25 notify lci 257 event 1
15:07:23: QLLC X25 notify lci 257 event 5
15:07:34: QLLC X25 notify lci 257 event 3 Caller 00407116 Caller 00400BD2
15:07:35: QLLC X25 notify lci 257 event 4
```

Figure 2-94 Sample Debug QLLC X25 Output

Table 2-51 describes fields of output that appear in Figure 2-94 follow.

Table 2-51 Debug QLLC X.25 Field Descriptions

Field	Description
15:07:23	Shows the time of day.
QLLC X25 notify 257	Indicates this is a QLLC X25 message.
event *n*	Indicates the type of event, *n*. Values for *n* can be as follows: 1 – Circuit is cleared 2 – Circuit has been reset 3 – Circuit is connected 4 – Circuit congestion has cleared 5 – Circuit has been deleted

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

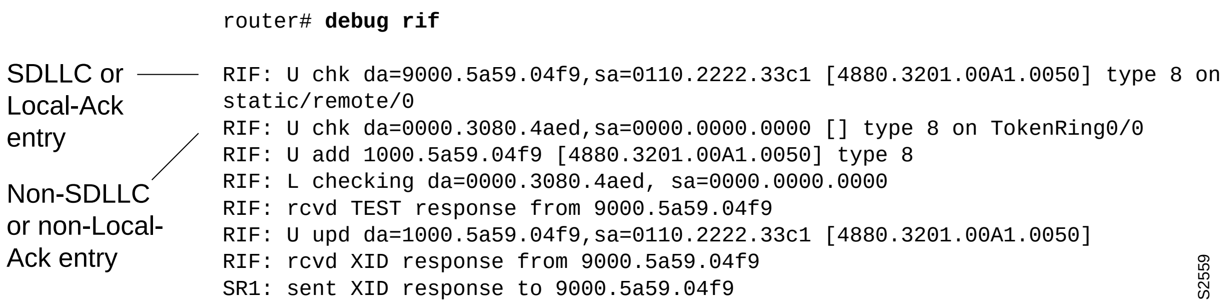
EXEC

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 2-95 shows sample **debug rif** output.



```
router# debug rif

RIF: U chk da=9000.5a59.04f9,sa=0110.2222.33c1 [4880.3201.00A1.0050] type 8 on
static/remote/0
RIF: U chk da=0000.3080.4aed,sa=0000.0000.0000 [] type 8 on TokenRing0/0
RIF: U add 1000.5a59.04f9 [4880.3201.00A1.0050] type 8
RIF: L checking da=0000.3080.4aed, sa=0000.0000.0000
RIF: rcvd TEST response from 9000.5a59.04f9
RIF: U upd da=1000.5a59.04f9,sa=0110.2222.33c1 [4880.3201.00A1.0050]
RIF: rcvd XID response from 9000.5a59.04f9
SR1: sent XID response to 9000.5a59.04f9
```

Figure 2-95 Sample Debug RIF Output

Explanations for representative lines of **debug rif** output in Figure 2-95 follow.

The first line of output is an example of a RIF entry for an interface configured for SDLLC or Local-Ack. Table 2-52 describes significant fields shown in this line of **debug rif** output.

Table 2-52 Debug RIF Field Descriptions—Part 1

Field	Description
RIF:	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	This route was learned from a real Token Ring port, in contrast to a virtual ring.

The following line of output 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 following line 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 following line shows that a RIF cache lookup operation has taken place:

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

The following line 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 following line 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 following line shows that an XID response from this address was inserted into the RIF cache:

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

The following line shows that the router sent an XID response to this address:

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

Table 2-53 explains the other possible lines of **debug rif** output.

Table 2-53 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 2-96 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
```

Figure 2-96 Sample Debug SDLC Output

Explanations for individual lines of output from Figure 2-96 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 2-54 describes the fields in this line of output.

Table 2-54 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).
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

Field	Description
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 2-54, output for a frame input event also includes two additional fields, as described in Table 2-55.

Table 2-55 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 2-56 describes the fields in this line of output.

Table 2-56 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 [*number*]
no debug sdlc local-ack [*number*]

Syntax Description

number	(Optional) Frame type that you want to monitor. Refer to the "Usage Guidelines" section.

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 2-57 defines these bit flags.

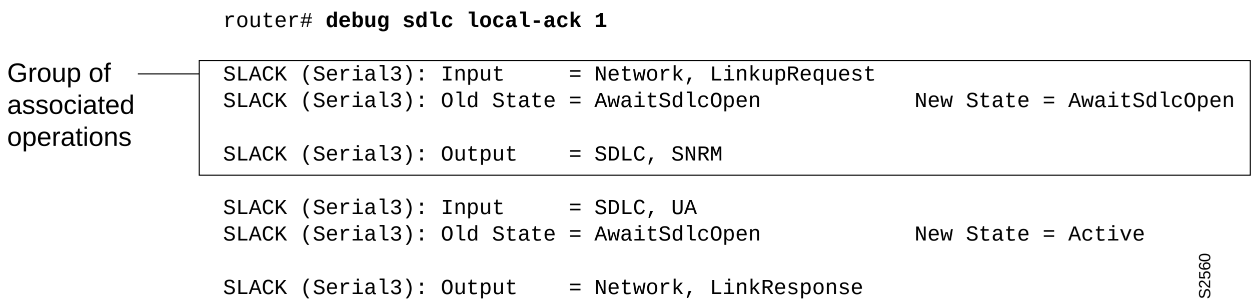
Table 2-57 Debug SDLC Local-Ack Debugging Levels

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 use this command by itself, rather than in conjunction with other debugging commands.

Sample Display

Figure 2-97 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
```

Figure 2-97 Sample Debug SDLC Local-Ack Output

Explanations for individual lines of output from Figure 2-97 follow.

The first line shows the input to the SDLC local acknowledgment state machine:

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

Table 2-58 describes the fields in this line of output.

Table 2-58 Debug SDLC Local-Ack Field Descriptions

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

The second line 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 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 2-98 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
```

Figure 2-98 Sample Debug SDLLC Output

Table 2-59 describes significant fields shown in Figure 2-98:

Table 2-59 Debug SDLLC Field Descriptions

Field	Description
rx	Router receives message from the FEP.
explorer 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 indicates that an explorer frame response was 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 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 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 is the SABME response to the XID command previously sent by the router to the FEP:

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