

debug clns esis events

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

[no] debug clns esis events

Sample Display

Figure 2-47 shows sample **debug clns esis events** output.

Figure 2-47 Sample Debug CLNS ESIS Events Output

```
Router# debug clns esis events

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

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

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

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

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

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

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

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

debug clns esis packets

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

[no] debug clns esis packets

Sample Display

Figure 2-48 shows sample **debug clns esis packets** output.

Figure 2-48 Sample Debug CLNS ESIS Packets Output

```
Router# debug clns esis packets

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

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

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

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

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

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

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

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

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

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

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

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

debug clns events

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

[no] debug clns events

Sample Display

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

Figure 2-49 Sample Debug CLNS Events Output

```
Router# debug clns events

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

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

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

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

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

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

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

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

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

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

debug clns igrp packets

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

[no] debug clns igrp packets

Sample Display

Figure 2-50 shows sample **debug clns igrp packets** output.

Figure 2-50 Sample Debug CLNS IGRP Packets Output

```
Router# debug clns igrp packets

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

Explanations for individual lines of output from Figure 2-50 follow.

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

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

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

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

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

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

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

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

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

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

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

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

debug clns packet

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

[no] debug clns packet

Sample Display

Figure 2-51 shows sample **debug clns packet** output.

Figure 2-51 Sample Debug CLNS Packet Output

```
Router# debug clns packet

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

Explanations for individual lines of output from Figure 2-51 follow.

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

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

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

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

debug clns routing

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

[no] debug clns routing

Sample Display

Figure 2-52 shows sample **debug clns routing** output.

Figure 2-52 Sample Debug CLNS Routing Output

```
Router# debug clns routing

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

Explanations for individual lines of output from Figure 2-52 follow.

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

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

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

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

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

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

debug cls message

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

[no] debug cls message

Usage Guidelines

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

Sample Display

Figure 2-53 shows sample **debug cls message** output. For example, *CLS-->DLU* indicates the direction of the flow that is described by the status. From CLS to DLU, a request was established to the connection end point. The header length is 48 bytes, and the data size is 104 bytes.

Figure 2-53 Sample Debug CLS Message Output

```
Router# debug cls message

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

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

Related Commands

debug fras error
debug fras message
debug fras state

debug cls vdlc

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

[no] debug cls vdlc

Usage Guidelines



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

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

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

Sample Displays

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

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

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

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

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

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

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

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

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

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

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

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

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

```

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

```

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

```

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

```

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

```

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

```

The following messages show the data flow:

```

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

```

Related Commands

debug cls message

debug dlsw core message

debug compress

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

[no] debug compress

Sample Display

Figure 2-54 shows sample **debug compress** output.

Figure 2-54 Sample Debug Compress Output

```
Router# debug compress

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

Table 2-25 describes significant **debug compress** output fields.

Table 2-25 Debug Compress Field Descriptions

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

debug confmodem

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

[no] **debug confmodem**

Usage Guidelines

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

Sample Display

Figure 2-55 shows sample **debug confmodem** output. In the first three lines, the router is searching for a speed at which it can communicate with the modem. The remaining lines show the actual sending of the modem command.

Figure 2-55 Sample Debug Configuration Modem Output

```
Router# debug confmodem

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

debug cpp event

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

[no] **debug cpp event**

Usage Guidelines

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

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

Sample Displays

One or more of the messages shown in Table 2-26 appear when you use the **debug cpp event** command. Each message begins with the short name of the interface the event occurred on (for example, *SERIAL0:1* or *BRI0:1*) and might contain one or more packet sequence numbers or remote site names.

Table 2-26 Debug CPP Event Messages

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

Related Commands

debug cpp negotiation

debug cpp packet

debug cpp negotiation

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

[no] debug cpp negotiation

Usage Guidelines

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

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

Sample Display

Figure 2-56 shows sample **debug cpp negotiation** output. In this example, a sample connection is shown.

Figure 2-56 Sample Debug CPP Negotiation Output

```
Router# debug cpp negotiation

%LINK-3-UPDOWN: Interface BRI0: B-Channel 2, changed state to down
%LINK-3-UPDOWN: Interface BRI0, changed state to up
%SYS-5-CONFIG_I: Configured from console by console
%LINK-3-UPDOWN: Interface BRI0: B-Channel 1, changed state to up
BR0:1:(I) NEG packet - len 77
  attempting proto:2
  ether id:0040.f902.c7b4
  port 1 number:5559876
  port 2 number:5559876
  origination port:1
  remote name:ber19
  password is correct
```

Table 2-27 shows describes the fields and messages shown in Figure 2-56.

Table 2-27 Debug CPP Negotiation Field Descriptions

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

Related Commands

debug cpp event

debug cpp packet

debug cpp packet

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

[no] debug cpp packet

Usage Guidelines

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

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

Sample Display

Figure 2-57 shows sample **debug cpp packet** output. This example shows the interface name, packet type, packet size, and the hexadecimal values of the packet.

Figure 2-57 Sample Debug CPP Packet Output

```
Router# debug cpp packet

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

Related Commands

debug cpp event

debug cpp negotiation

debug crypto key-exchange

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

[no] debug crypto key-exchange

Usage Guidelines

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

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

Sample Displays

Figure 2-58 and Figure 2-59 show sample **debug crypto key-exchange** output. Figure 2-58 shows output from the initiating router in a key exchange. Figure 2-59 shows output from the passive router in a key exchange. The number of bytes received should match the number of bytes sent from the initiating side, although the number of messages can be different.

Figure 2-58 Sample Debug Crypto Key-Exchange Output—Initiating Router

```
Router# debug crypto key-exchange

CRYPTO-KE: Sent 4 bytes.
CRYPTO-KE: Sent 2 bytes.
CRYPTO-KE: Sent 2 bytes.
CRYPTO-KE: Sent 2 bytes.
CRYPTO-KE: Sent 64 bytes.
```

Figure 2-59 Sample Debug Crypto Key-Exchange Output—Passive Router

```
Router# debug crypto key-exchange

CRYPTO-KE: Received 4 bytes.
CRYPTO-KE: Received 2 bytes.
CRYPTO-KE: Received 2 bytes.
CRYPTO-KE: Received 2 bytes.
CRYPTO-KE: Received 49 bytes.
CRYPTO-KE: Received 15 bytes.
```

Related Command

debug crypto sesgmt

debug crypto sesmgmt

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

[no] debug crypto sesmgmt

Usage Guidelines

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

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

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

Sample Displays

Figure 2-60 and Figure 2-61 show sample **debug crypto sesmgmt** output. Figure 2-60 shows messages from a router that initiates a successful connection. Figure 2-61 shows the messages from a router that receives a connection.

Figure 2-60 Sample Debug Crypto Sesmgmt Output—Initiating Router

```
Router# debug crypto sesmgmt

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

Figure 2-61 Sample Debug Crypto Sesmgmt Output—Receiving Router

```
Router# debug crypto sesmgmt

CRYPTO: Dequeued a message: CIM
CRYPTO: Verify done. Status=OK
CRYPTO: DH gen phase 1 status for conn_id 1 slot 0:OK
CRYPTO: DH gen phase 2 status for conn_id 1 slot 0:OK
CRYPTO: Signing done. Status:OK
```

debug crypto sesmgmt

```
CRYPTO: ICMP message sent: s=172.21.114.162, d=172.21.114.163
CRYPTO-SDU: act_on_nnc_req: NNC Echo Reply sent
CRYPTO: Create encryption key for conn_id 1 slot 0:OK
CRYPTO: Replacing -2 in crypto maps with 1 (slot 0)
CRYPTO: Dequeued a message: CCM
CRYPTO: Verify done. Status=OK
```

Related Command

debug crypto key-exchange

debug decnet adj

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

[no] debug decnet adj

Sample Display

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

Figure 2-62 Sample Debug DECnet Adj Output

```
Router# debug decnet adj

DNET-ADJ: Level 1 hello from 1.3
DNET-ADJ: sending hellos
DNET-ADJ: Sending hellos to all routers on interface Ethernet0, blksize 1498
DNET-ADJ: Level 1 hello from 1.3
DNET-ADJ: 1.5 adjacency initializing
DNET-ADJ: sending triggered hellos
DNET-ADJ: Sending hellos to all routers on interface Ethernet0, blksize 1498
DNET-ADJ: Level 1 hello from 1.3
DNET-ADJ: 1.5 adjacency up
DNET-ADJ: Level 1 hello from 1.5
DNET-ADJ: 1.5 adjacency down, listener timeout
```

Explanations for representative lines of output in Figure 2-62 follow.

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

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

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

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

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

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

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

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

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

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

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

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

debug decnet connects

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

[no] debug decnet connects

Usage Guidelines

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

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

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

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

Sample Display

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

Figure 2-63 Sample Debug DECnet Connects Output

```
Router# debug decnet connects

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

Table 2-28 describes significant fields shown in Figure 2-63.

Table 2-28 Debug DECnet Connects Field Descriptions

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

Table 2-28 Debug DECnet Connects Field Descriptions (Continued)

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

debug decnet events

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

[no] debug decnet events

Sample Display

Figure 2-64 shows sample **debug decnet events** output.

Figure 2-64 Sample Debug DECnet Events Output

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

Explanations for representative lines of output in Figure 2-64 follow.

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

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

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

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

debug decnet packet

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

[no] debug decnet packet

Sample Display

Figure 2-65 shows sample **debug decnet packet** output.

Figure 2-65 Sample Debug DECnet Packet Output

```
Router# debug decnet packet

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

Explanations for individual lines of output from Figure 2-65 follow.

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

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

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

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

debug decnet routing

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

[no] debug decnet routing

Sample Display

Figure 2-66 shows sample **debug decnet routing** output.

Figure 2-66 Sample Debug DECnet Routing Output

```
Router# debug decnet routing

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

Explanations for individual lines of output from Figure 2-66 follow.

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

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

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

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

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

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

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

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

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

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

debug dialer events

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

[no] debug dialer events

Sample Displays

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

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

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

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

Most messages are self-explanatory; however, messages that may need some explanation are described in Table 2-29.

Table 2-29 General Debug Dialer Events Message Descriptions

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

Table 2-30 describes the messages that the **debug dialer events** command can generate for a serial interface used as a V.25*bis* dialer for dial-on-demand routing (DDR).

Table 2-30 Debug Dialer Events Message Descriptions for DDR

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

Related Command
debug dialer packets

debug dialer packets

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

[no] debug dialer packets

Usage Guidelines

Most debug dialer packet messages are self-explanatory.

Sample Display

Figure 2-67 shows sample **debug dialer packets** output. The following message shows the interface type, the type of packet (protocol) being sent, the source and destination addresses, the size of the packet, and the default action for the packet (in this example, *permit*).

Figure 2-67 Sample Debug Dialer Packets Output

```
Router# debug dialer packets

BRI0: ip (s=10.1.1.8, d=10.1.1.1), 100 bytes, interesting (ip PERMIT)
```

Related Command

debug dialer events

debug dlsw

Use the **debug dlsw** EXEC command to enable debugging of DLSw+. The **no** form of this command disables debugging output.

```
[no] debug dlsw [border-peers [interface interface | ip address ip address] | core
[flow-control | messages | state | xid] [circuit number] | local-circuit circuit number | peers
[interface interface [fast-errors | fast-paks] | ip address ip address [fast-errors |
fast-paks | fst-seq | udp]] | reachability [error | verbose] [sna | netbios]]
```

Syntax Description

border-peers	(Optional) Enables debugging output for border peer events.
interface <i>interface</i>	(Optional) Specifies a remote peer to debug by a direct interface.
ip address <i>ip address</i>	(Optional) Specifies a remote peer to debug by its IP address.
core	(Optional) Enables debugging output for DLSw core events.
flow-control	(Optional) Enables debugging output for congestion in the WAN or at the remote end station.
messages	(Optional) Enables debugging output of core messages—specific packets received by DLSw either from one of its peers or from a local medium via the Cisco link services interface.
state	(Optional) Enables debugging output for state changes on the circuit.
xid	(Optional) Enables debugging output for the exchange identification-state machine.
<i>circuit-number</i>	(Optional) Specifies the circuit for which you want core debugging output to reduce the of output.
local-circuit <i>circuit number</i>	(Optional) Enables debugging output for circuits performing local conversion. Local conversion occurs when both the input and output data-link connections are on the same local peer and no remote peer exists.
peers	(Optional) Enables debugging output for peer events.
fast-errors	(Optional) Debugs errors for fast-switched packets.
fast-paks	(Optional) Debugs fast-switched packets.
fst-seq	(Optional) Debug FST sequence numbers on fast switched packets.
udp	(Optional) Debug UDP packets.
reachability	(Optional) Enables debugging output for reachability events (explorer traffic). If no options are specified, event-level information is displayed for all protocols.

error verbose	(Optional) Specifies how much reachability information you want displayed. The verbose keyword displays everything, including errors and events. The error keyword displays error information only. If no option is specified, event-level information is displayed.
sna netbios	(Optional) Specifies that reachability information be displayed for only SNA or NetBIOS protocols. If no option is specified, information for all protocols is displayed.

Usage Guidelines

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

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

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

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

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

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

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

Sample Displays

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

Sample Debug DLSW Peer Messages

The following example enables UDP packet debugging for a specific remote peer:

```
Router# debug dlsw peer ip-address 1.1.1.6 udp
```

The following message is sample output from the **debug dlsw border-peers** command:

```
*Mar 10 17:39:56: CSM: delete group mac cache for group 0
*Mar 10 17:39:56: CSM: delete group name cache for group 0
*Mar 10 17:40:19: CSM: update group cache for mac 0000.3072.1070, group 10
*Mar 10 17:40:22: DLSw: send_to_group_members(): copy to peer 10.19.32.5
```

The following message is from a router that initiated a TCP connection:

```
DLSw: START-TPFSM (peer 10.3.8.7(2065)): event:ADMIN-OPEN CONNECTION state:DISCONN
DLSw: dtp_action_a() attempting to connect peer 10.3.8.7(2065)
DLSw: END-TPFSM (peer 10.3.8.7(2065)): state:DISCONN->WAIT_WR
DLSw: Async Open Callback 10.3.8.7(2065) -> 11002
DLSw: START-TPFSM (peer 10.3.8.7(2065)): event:TCP-WR PIPE OPENED state:WAIT_WR
DLSw: dtp_action_f() start read open timer for peer 10.3.8.7(2065)
DLSw: END-TPFSM (peer 10.3.8.7(2065)): state:WAIT_WR->WAIT_RD
DLSw: passive open 10.3.8.7(11004) -> 2065
DLSw: START-TPFSM (peer 10.3.8.7(2065)): event:TCP-RD PIPE OPENED state:WAIT_RD
DLSw: dtp_action_g() read pipe opened for peer 10.3.8.7(2065)
DLSw: CapExId Msg sent to peer 10.3.8.7(2065)
DLSw: END-TPFSM (peer 10.3.8.7(2065)): state:WAIT_RD->WAIT_CAP
DLSw: START-TPFSM (peer 10.3.8.7(2065)): event:SSP-CAP MSG RCVD state:WAIT_CAP
DLSw: dtp_action_j() cap msg rcvd from peer 10.3.8.7(2065)
DLSw: Recv CapExId Msg from peer 10.3.8.7(2065)
DLSw: Pos CapExResp sent to peer 10.3.8.7(2065)
DLSw: END-TPFSM (peer 10.3.8.7(2065)): state:WAIT_CAP->WAIT_CAP
DLSw: START-TPFSM (peer 10.3.8.7(2065)): event:SSP-CAP MSG RCVD state:WAIT_CAP
DLSw: dtp_action_j() cap msg rcvd from peer 10.3.8.7(2065)
DLSw: Recv CapExPosRsp Msg from peer 10.3.8.7(2065)
DLSw: END-TPFSM (peer 10.3.8.7(2065)): state:WAIT_CAP->WAIT_CAP
DLSw: Processing delayed event:SSP-CAP EXCHANGED - prev state:WAIT_CAP
DLSw: START-TPFSM (peer 10.3.8.7(2065)): event:SSP-CAP EXCHANGED state:WAIT_CAP
DLSw: dtp_action_k() cap xchgd for peer 10.3.8.7(2065)
DLSw: closing read pipe tcp connection for peer 10.3.8.7(2065)
DLSw: END-TPFSM (peer 10.3.8.7(2065)): state:WAIT_CAP->PCONN_WT
DLSw: Processing delayed event:TCP-PEER CONNECTED - prev state:PCONN_WT
DLSw: START-TPFSM (peer 10.3.8.7(2065)): event:TCP-PEER CONNECTED state:PCONN_WT
DLSw: dtp_action_m() peer connected for peer 10.3.8.7(2065)
DLSw: END-TPFSM (peer 10.3.8.7(2065)): state:PCONN_WT->CONNECT
DLSw: START-TPFSM (peer 10.3.8.7(2065)): event:CORE-ADD CIRCUIT state:CONNECT
DLSw: dtp_action_u(), peer add circuit for peer 10.3.8.7(2065)
DLSw: END-TPFSM (peer 10.3.8.7(2065)): state:CONNECT->CONNECT
```

The following message is from a router that received a TCP connection:

```
DLSw: passive open 10.10.10.4(11002) -> 2065
DLSw: START-TPFSM (peer 10.10.10.4(2065)): event:TCP-RD PIPE OPENED state:DISCONN
DLSw: dtp_action_c() opening write pipe for peer 10.10.10.4(2065)
DLSw: END-TPFSM (peer 10.10.10.4(2065)): state:DISCONN->WWR_RDOP
DLSw: Async Open Callback 10.10.10.4(2065) -> 11004
DLSw: START-TPFSM (peer 10.10.10.4(2065)): event:TCP-WR PIPE OPENED state:WWR_RDOP
DLSw: dtp_action_i() write pipe opened for peer 10.10.10.4(2065)
DLSw: CapExId Msg sent to peer 10.10.10.4(2065)
DLSw: END-TPFSM (peer 10.10.10.4(2065)): state:WWR_RDOP->WAIT_CAP
DLSw: START-TPFSM (peer 10.10.10.4(2065)): event:SSP-CAP MSG RCVD state:WAIT_CAP
DLSw: dtp_action_j() cap msg rcvd from peer 10.10.10.4(2065)
DLSw: Recv CapExId Msg from peer 10.10.10.4(2065)
DLSw: Pos CapExResp sent to peer 10.10.10.4(2065)
DLSw: END-TPFSM (peer 10.10.10.4(2065)): state:WAIT_CAP->WAIT_CAP
DLSw: START-TPFSM (peer 10.10.10.4(2065)): event:SSP-CAP MSG RCVD state:WAIT_CAP
DLSw: dtp_action_j() cap msg rcvd from peer 10.10.10.4(2065)
DLSw: Recv CapExPosRsp Msg from peer 10.10.10.4(2065)
DLSw: END-TPFSM (peer 10.10.10.4(2065)): state:WAIT_CAP->WAIT_CAP
DLSw: Processing delayed event:SSP-CAP EXCHANGED - prev state:WAIT_CAP
DLSw: START-TPFSM (peer 10.10.10.4(2065)): event:SSP-CAP EXCHANGED state:WAIT_CAP
DLSw: dtp_action_k() cap xchgd for peer 10.10.10.4(2065)
DLSw: END-TPFSM (peer 10.10.10.4(2065)): state:WAIT_CAP->PCONN_WT
DLSw: dlsw_tcpd_fini() for peer 10.10.10.4(2065)
DLSw: dlsw_tcpd_fini() closing write pipe for peer 10.10.10.4
DLSw: START-TPFSM (peer 10.10.10.4(2065)): event:TCP-CLOSE WR PIPE state:PCONN_WT
DLSw: dtp_action_l() close write pipe for peer 10.10.10.4(2065)
DLSw: closing write pipe tcp connection for peer 10.10.10.4(2065)
```

```

DLsw: END-TPFSM (peer 10.10.10.4(2065)): state:PCONN_WT->PCONN_WT
DLsw: Processing delayed event:TCP-PEER CONNECTED - prev state:PCONN_WT
DLsw: START-TPFSM (peer 10.10.10.4(2065)): event:TCP-PEER CONNECTED state:PCONN_WT
DLsw: dtp_action_m() peer connected for peer 10.10.10.4(2065)
DLsw: END-TPFSM (peer 10.10.10.4(2065)): state:PCONN_WT->CONNECT
DLsw: START-TPFSM (peer 10.10.10.4(2065)): event:CORE-ADD CIRCUIT state:CONNECT
DLsw: dtp_action_u(), peer add circuit for peer 10.10.10.4(2065)
DLsw: END-TPFSM (peer 10.10.10.4(2065)): state:CONNECT->CONNECT

```

The following message is from a router that initiated an FST connection:

```

DLsw: START-FSTPFMSM (peer 10.10.10.4(0)): event:ADMIN-OPEN CONNECTION state:DISCONN
DLsw: dfstp_action_a() attempting to connect peer 10.10.10.4(0)
DLsw: Connection opened for peer 10.10.10.4(0)
DLsw: CapExId Msg sent to peer 10.10.10.4(0)
DLsw: END-FSTPFMSM (peer 10.10.10.4(0)): state:DISCONN->WAIT_CAP
DLsw: START-FSTPFMSM (peer 10.10.10.4(0)): event:SSP-CAP MSG RCVD state:WAIT_CAP
DLsw: dfstp_action_e() cap msg rcvd for peer 10.10.10.4(0)
DLsw: Recv CapExPosRsp Msg from peer 10.10.10.4(0)
DLsw: END-FSTPFMSM (peer 10.10.10.4(0)): state:WAIT_CAP->WAIT_CAP
DLsw: START-FSTPFMSM (peer 10.10.10.4(0)): event:SSP-CAP MSG RCVD state:WAIT_CAP
DLsw: dfstp_action_e() cap msg rcvd for peer 10.10.10.4(0)
DLsw: Recv CapExId Msg from peer 10.10.10.4(0)
DLsw: Pos CapExResp sent to peer 10.10.10.4(0)
DLsw: END-FSTPFMSM (peer 10.10.10.4(0)): state:WAIT_CAP->WAIT_CAP
DLsw: Processing delayed event:SSP-CAP EXCHANGED - prev state:WAIT_CAP
DLsw: START-FSTPFMSM (peer 10.10.10.4(0)): event:SSP-CAP EXCHANGED state:WAIT_CAP
DLsw: dfstp_action_f() cap xchged for peer 10.10.10.4(0)
DLsw: END-FSTPFMSM (peer 10.10.10.4(0)): state:WAIT_CAP->CONNECT

```

The following message is from a router that received an FST connection:

```

DLsw: START-FSTPFMSM (peer 10.3.8.7(0)): event:SSP-CAP MSG RCVD state:DISCONN
DLsw: dfstp_action_c() cap msg rcvd for peer 10.3.8.7(0)
DLsw: Recv CapExId Msg from peer 10.3.8.7(0)
DLsw: Pos CapExResp sent to peer 10.3.8.7(0)
DLsw: CapExId Msg sent to peer 10.3.8.7(0)
DLsw: END-FSTPFMSM (peer 10.3.8.7(0)): state:DISCONN->WAIT_CAP
DLsw: START-FSTPFMSM (peer 10.3.8.7(0)): event:SSP-CAP MSG RCVD state:WAIT_CAP
DLsw: dfstp_action_e() cap msg rcvd for peer 10.3.8.7(0)
DLsw: Recv CapExPosRsp Msg from peer 10.3.8.7(0)
DLsw: END-FSTPFMSM (peer 10.3.8.7(0)): state:WAIT_CAP->WAIT_CAP
DLsw: Processing delayed event:SSP-CAP EXCHANGED - prev state:WAIT_CAP
DLsw: START-FSTPFMSM (peer 10.3.8.7(0)): event:SSP-CAP EXCHANGED state:WAIT_CAP
DLsw: dfstp_action_f() cap xchged for peer 10.3.8.7(0)
DLsw: END-FSTPFMSM (peer 10.3.8.7(0)): state:WAIT_CAP->CONNECT

```

The following message is from a router that initiated an LLC2 connection:

```

DLsw-LLC2: Sending enable port ; port no : 0
          PEER-DISP Sent : CLSI Msg : ENABLE.Req  dlen: 20
DLsw: Peer Received : CLSI Msg : ENABLE.Cfm CLS_OK dlen: 20
DLsw-LLC2 : Sending activate sap for Serial1 - port_id = 887C3C
          port_type = 7 dgra(UsapID) = 952458
          PEER-DISP Sent : CLSI Msg : ACTIVATE_SAP.Req  dlen: 60
DLsw: Peer Received : CLSI Msg : ACTIVATE_SAP.Cfm CLS_OK dlen: 60
DLsw Got ActSapcnf back for Serial1 - port_id = 8978204, port_type = 7, psap_id = 0

DLsw: START-LLC2PFMSM (peer on interface Serial1): event:ADMIN-OPEN CONNECTION
state:DISCONN
DLsw: dllc2p_action_a() attempting to connect peer on interface Serial1
          PEER-DISP Sent : CLSI Msg : REQ_OPNSTN.Req  dlen: 106
DLsw: END-LLC2PFMSM (peer on interface Serial1): state:DISCONN->ROS_SENT

DLsw: Peer Received : CLSI Msg : REQ_OPNSTN.Cfm CLS_OK dlen: 106

```

```

DLsw: START-LLC2PFSM (peer on interface Serial1): event:CLS-REQOPNSTN.CNF state:ROS_SENT
DLsw: dllc2p_action_c()
  PEER-DISP Sent : CLSI Msg : CONNECT.Reg  dlen: 16
DLsw: END-LLC2PFSM (peer on interface Serial1): state:ROS_SENT->CON_PEND

DLsw: Peer Received : CLSI Msg : CONNECT.Cfm CLS_OK dlen: 28
DLsw: START-LLC2PFSM (peer on interface Serial1): event:CLS-CONNECT.CNF state:CON_PEND
DLsw: dllc2p_action_e() send capabilities to peer on interface Serial1
  PEER-DISP Sent : CLSI Msg : SIGNAL_STN.Reg  dlen: 8
  PEER-DISP Sent : CLSI Msg : DATA.Reg  dlen: 418
DLsw: CapExId Msg sent to peer on interface Serial1
DLsw: END-LLC2PFSM (peer on interface Serial1): state:CON_PEND->WAIT_CAP

DLsw: Peer Received : CLSI Msg : DATA.Ind  dlen: 418
DLsw: START-LLC2PFSM (peer on interface Serial1): event:SSP-CAP MSG RCVD state:WAIT_CAP
DLsw: dllc2p_action_k() cap msg rcvd for peer on interface Serial1
DLsw: Recv CapExId Msg from peer on interface Serial1
  PEER-DISP Sent : CLSI Msg : DATA.Reg  dlen: 96
DLsw: Pos CapExResp sent to peer on interface Serial1
DLsw: END-LLC2PFSM (peer on interface Serial1): state:WAIT_CAP->WAIT_CAP

DLsw: Peer Received : CLSI Msg : DATA.Ind  dlen: 96
DLsw: START-LLC2PFSM (peer on interface Serial1): event:SSP-CAP MSG RCVD state:WAIT_CAP
DLsw: dllc2p_action_k() cap msg rcvd for peer on interface Serial1
DLsw: Recv CapExPosRsp Msg from peer on interface Serial1
DLsw: END-LLC2PFSM (peer on interface Serial1): state:WAIT_CAP->WAIT_CAP

DLsw: Processing delayed event:SSP-CAP EXCHANGED - prev state:WAIT_CAP
DLsw: START-LLC2PFSM (peer on interface Serial1): event:SSP-CAP EXCHANGED state:WAIT_CAP
DLsw: dllc2p_action_l() cap xchged for peer on interface Serial1
DLsw: END-LLC2PFSM (peer on interface Serial1): state:WAIT_CAP->CONNECT

```

The following message is from a router that received an LLC2 connection:

```

DLsw-LLC2: Sending enable port ; port no : 0
  PEER-DISP Sent : CLSI Msg : ENABLE.Reg  dlen: 20
DLsw: Peer Received : CLSI Msg : ENABLE.Cfm CLS_OK dlen: 20
DLsw-LLC2 : Sending activate sap for Serial0 - port_id = 887C3C
  port_type = 7 dgra(UsapID) = 93AB34
  PEER-DISP Sent : CLSI Msg : ACTIVATE_SAP.Reg  dlen: 60
DLsw: Peer Received : CLSI Msg : ACTIVATE_SAP.Cfm CLS_OK dlen: 60
DLsw Got ActSapcnf back for Serial0 - port_id = 8944700, port_type = 7, psap_id = 0

DLsw: Peer Received : CLSI Msg : CONECT_STN.Ind  dlen: 39
DLsw: START-LLC2PFSM (peer on interface Serial0): event:CLS-CONNECT_STN.IND
state:DISCONN
DLsw: dllc2p_action_s() conn_stn for peer on interface Serial0
  PEER-DISP Sent : CLSI Msg : REQ_OPNSTN.Reg  dlen: 106
DLsw: END-LLC2PFSM (peer on interface Serial0): state:DISCONN->CONS_PEND

DLsw: Peer Received : CLSI Msg : REQ_OPNSTN.Cfm CLS_OK dlen: 106
DLsw: START-LLC2PFSM (peer on interface Serial0): event:CLS-REQOPNSTN.CNF
state:CONS_PEND
DLsw: dllc2p_action_h() send capabilities to peer on interface Serial0
  PEER-DISP Sent : CLSI Msg : CONNECT.Rsp  dlen: 20
  PEER-DISP Sent : CLSI Msg : DATA.Reg  dlen: 418
DLsw: CapExId Msg sent to peer on interface Serial0
DLsw: END-LLC2PFSM (peer on interface Serial0): state:CONS_PEND->WAIT_CAP

DLsw: Peer Received : CLSI Msg : CONNECTED.Ind  dlen: 8
DLsw: START-LLC2PFSM (peer on interface Serial0): event:CLS-CONNECTED.IND state:WAIT_CAP
DLsw: END-LLC2PFSM (peer on interface Serial0): state:WAIT_CAP->WAIT_CAP

DLsw: Peer Received : CLSI Msg : DATA.Ind  dlen: 418
DLsw: START-LLC2PFSM (peer on interface Serial0): event:SSP-CAP MSG RCVD state:WAIT_CAP

```

```

DLSw: dllc2p_action_k() cap msg rcvd for peer on interface Serial0
DLSw: Recv CapExId Msg from peer on interface Serial0
  PEER-DISP Sent : CLSI Msg : DATA.Req  dlen: 96
DLSw: Pos CapExResp sent to peer on interface Serial0
DLSw: END-LLC2PFSM (peer on interface Serial0): state:WAIT_CAP->WAIT_CAP

DLSw: Peer Received : CLSI Msg : DATA.Ind  dlen: 96
DLSw: START-LLC2PFSM (peer on interface Serial0): event:SSP-CAP MSG RCVD state:WAIT_CAP
DLSw: dllc2p_action_k() cap msg rcvd for peer on interface Serial0
DLSw: Recv CapExPosRsp Msg from peer on interface Serial0
DLSw: END-LLC2PFSM (peer on interface Serial0): state:WAIT_CAP->WAIT_CAP

DLSw: Processing delayed event:SSP-CAP EXCHANGED - prev state:WAIT_CAP
DLSw: START-LLC2PFSM (peer on interface Serial0): event:SSP-CAP EXCHANGED state:WAIT_CAP
DLSw: dllc2p_action_l() cap xchgd for peer on interface Serial0
DLSw: END-LLC2PFSM (peer on interface Serial0): state:WAIT_CAP->CONNECT

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

23:13:22: action_a() attempting to connect peer 172.20.100.1(2065)
23:13:22: DLSw: CONN: peer 172.20.100.1 open failed, rejected [9]
23:13:22: action_a() retries: 8 next conn time: 861232504
23:13:52: action_a() attempting to connect peer 172.20.100.1(2065)
23:13:52: DLSw: CONN: peer 172.20.100.1 open failed, rejected [9]
23:13:52: action_a() retries: 9 next conn time: 861292536

```

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

```

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

```

The following messages show a peer connection opening with no errors or abnormal events:

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

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

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

Sample Debug DLSw Reachability Messages

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

- Reachability is stored in cache. DLSw+ maintains two reachability caches: one for MAC addresses and one for NetBIOS names. Depending on how long entries have been in the cache, they are either fresh or stale.
- If a router has a fresh entry in the cache for a certain resource, it answers a locate request for that resource without verifying that it is still available. A locate request is typically a TEST frame for MAC addresses or a FIND_NAME_QUERY for NetBIOS.
- If a router has a stale entry in the cache for a certain resource, it verifies that the entry is still valid before answering a locate request for the resource by sending a frame to the resource's last known location and waits for a resource. If the entry is a REMOTE entry, the router sends a CUR_ex frame to the remote peer to verify. If the entry is a LOCAL entry, it sends either a TEST frame or a NetBIOS FIND_NAME_QUERY on the appropriate local port.
- By default, all reachability cache entries remain fresh for 4 minutes after they are learned. For MAC addresses, you can change this time with the **dlsw timer sna-verify-interval** command. For NetBIOS names, you can change this with the **dlsw timer netbios-verify-interval** command.
- By default, all reachability cache entries age out of the cache 16 minutes after they are learned. For MAC addresses, you can change this time with the **dlsw timer sna-cache-timeout** command. For NetBIOS names, you can change the time with the **dlsw timer netbios-cache-timeout** command.

Table 2-31 describes the debug output indicating that the dlsw router received an SSP message that is flow controlled and should be counted against the sender's window.

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

Table 2-31 Debug Output Descriptions

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

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

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

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

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

The following messages show that the reachability cache is cleared:

```
Router# clear dlsw rea

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

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

```
23:45:11: CSM: core_to_csm CLSI_MSG_PROC - port_id 5EFBB4
23:45:11: CSM: 0800.5a30.7a9b passes local mac excl. filter
23:45:11: CSM: update local cache for mac 0800.5a30.7a9b, port 5EFBB4
23:45:11: CSM: update local cache for name MARIAN , port 5EFBB4
23:45:11: CSM: Received CLS_UDATA_STN from Core
23:45:11: CSM: Received netbios frame type A
23:45:11: CSM: Processing Name Query
23:45:11: CSM: Netbios Name Query: ws_status = 6
23:45:11: CSM: Write to peer 0 ok.
23:45:11: CSM: Freeing clsi message
23:45:11: CSM: core_to_csm CLSI_MSG_PROC - port_id 658AB4
23:45:11: CSM: 0800.5a30.7a9b passes local mac excl. filter
23:45:11: CSM: update local cache for mac 0800.5a30.7a9b, port 658AB4
23:45:11: CSM: update local cache for name MARIAN , port 658AB4
23:45:11: CSM: Received CLS_UDATA_STN from Core
23:45:11: CSM: Received netbios frame type A
23:45:11: CSM: Processing Name Query
23:45:11: CSM: Netbios Name Query: ws_status = 5
23:45:11: CSM: DLXNR_PEND match found.... drop name query
23:45:11: CSM: Freeing clsi message
23:45:12: CSM: core_to_csm CLSI_MSG_PROC - port_id 5EFBB4
23:45:12: CSM: 0800.5a30.7a9b passes local mac excl. filter
23:45:12: CSM: update local cache for mac 0800.5a30.7a9b, port 5EFBB4
23:45:12: CSM: update local cache for name MARIAN , port 5EFBB4
```

```

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

```

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

```

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

```

```

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

```

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

```

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

```

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

```

G41# show dlsw reachability

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

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

```

debug dspu activation

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

[no] debug dspu activation *[name]*

Syntax Description

name (Optional) A host or PU name designation.

Usage Guidelines

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

Sample Display

Figure 2-68 shows sample **debug dspu activation** output. Not all intermediate numbers are shown for the “activated” and “deactivated” logical unit (LU) address ranges.

Figure 2-68 Sample Debug DSPU Activation Output

```
Router# debug dspu activation

DSPU: LS HOST3745 connected
DSPU: PU HOST3745 activated
DSPU: LU HOST3745-2 activated
DSPU: LU HOST3745-3 activated
...
DSPU: LU HOST3745-253 activated
DSPU: LU HOST3745-254 activated

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

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

Table 2-32 Debug DSPU Activation Field Descriptions

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

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

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

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

debug dspu packet

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

[no] debug dspu packet [*name*]

Syntax Description

name (Optional) A host or PU name designation.

Usage Guidelines

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

Sample Display

Figure 2-69 shows sample **debug dspu packet** output.

Figure 2-69 Sample Debug DSPU Packet Output

```
Router# debug dspu packet

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

Table 2-33 describes significant fields in the output shown in Figure 2-69.

Table 2-33 Debug DSPU Packet Field Descriptions

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

Related Commands

- debug dspu activation**
- debug dspu state**
- debug dspu trace**

debug dspu state

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

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

Syntax Description

name (Optional) A host or PU name designation.

Usage Guidelines

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

Sample Display

Figure 2-70 shows sample **debug dspu state** output. Not all intermediate numbers are shown for the “activated” and “deactivated” logical unit (LU) address ranges.

Figure 2-70 Sample Debug DSPU State Output

```
Router# debug dspu state

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

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

Table 2-34 describes significant fields in the output shown in Figure 2-70.

Table 2-34 Debug DSPU State Field Descriptions

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

Table 2-34 Debug DSPU State Field Descriptions (Continued)

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

Related Commands

debug dspu activation

debug dspu packet

debug dspu trace

debug dspu trace

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

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

Syntax Description

name (Optional) A host or PU name designation.

Usage Guidelines

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

Sample Display

Figure 2-71 shows sample **debug dspu trace** output.

Figure 2-71 Sample Debug DSPU Trace Output

```
Router# debug dspu trace

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

Table 2-35 describes significant fields in the output shown in Figure 2-71.

Table 2-35 Debug DSPU Trace Field Descriptions

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

Table 2-35 Debug DSPU Trace Field Descriptions (Continued)

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

Related Commands

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

debug eigrp fsm

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

[no] debug eigrp fsm

Usage Guidelines

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

Sample Display

Figure 2-72 shows sample **debug eigrp fsm** output.

Figure 2-72 Sample Debug EIGRP FSM Output

```
Router# debug eigrp fsm

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

Explanations for individual lines of output from Figure 2-72 follow.

In the first line of Figure 2-72, DUAL stands for Diffusing Update ALgorithm. It is the basic mechanism within Enhanced IGRP that makes the routing decisions. The next three fields are the Internet address and mask of the destination network and the address through which the update was received. The metric field shows the metric stored in the routing table and the metric advertised by the neighbor sending the information. “Metric... inaccessible” usually means that the neighbor router no longer has a route to the destination, or the destination is in holddown.

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

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

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

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

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

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

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

debug eigrp packet

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

[no] debug eigrp packet

Usage Guidelines

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

Sample Display

Figure 2-73 shows sample **debug eigrp packet** output.

Figure 2-73 Sample Debug EIGRP Packet Output

```
Router# debug eigrp packet

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

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

Table 2-36 describes significant fields in the output shown in Figure 2-73.

Table 2-36 Debug EIGRP Packet Field Descriptions

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

Table 2-36 Debug EIGRP Packet Field Descriptions (Continued)

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

debug fddi smt-packets

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

```
[no] debug fddi smt-packets
```

Sample Display

Figure 2-74 shows sample **debug fddi smt-packets** output. In this example, an SMT frame has been output by Fiber Distributed Data Interface (FDDI) 1/0. The SMT frame is a next station addressing (NSA) neighbor information frame (NIF) request frame with the parameters as shown.

Figure 2-74 Sample Debug FDDI SMT Output

```
Router# debug fddi smt-packets

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

Table 2-37 describes the fields in the output shown in Figure 2-74.

Table 2-37 Debug FDDI SMT-Packets Field Descriptions

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

debug frame-relay

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

[no] debug frame-relay

Usage Guidelines

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

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

Sample Display

Figure 2-75 shows sample **debug frame-relay** output.

Figure 2-75 Sample Debug Frame-Relay Output

```
Router# debug frame-relay

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

Table 2-38 describes significant fields shown in Figure 2-75.

Table 2-38 Debug Frame-Relay Field Descriptions

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

Table 2-38 Debug Frame-Relay Field Descriptions (Continued)

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

debug frame-relay callcontrol

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

[no] debug frame-relay callcontrol

Usage Guidelines

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

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

Sample Display

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

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

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

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

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

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

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

Table 2-39 explains the possible call states.

Table 2-39 Frame Relay Switched Virtual Circuit (SVC) Call States

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

Table 2-39 Frame Relay Switched Virtual Circuit (SVC) Call States (Continued)

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

Related Commands

debug frame-relay

debug frame-relay networklayerinterface

debug frame-relay events

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

[no] **debug frame-relay events**

Usage Guidelines

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

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

Sample Display

Figure 2-76 shows sample **debug frame-relay events** output.

Figure 2-76 Sample Debug Frame-Relay Events Output

```
Router# debug frame-relay events

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

As Figure 2-76 shows, **debug frame-relay events** returns one specific message type. The first line, for example, indicates that IP address 172.16.170.26 sent a Frame Relay ARP reply; this packet was received as input on the Serial2 interface. The last field (126) is the data link connection identifier (DLCI) to use when communicating with the responding router.

debug frame-relay foresight

Use the **debug frame-relay foresight** EXEC command to observe Frame Relay traces relating to traffic shaping with router ForeSight enabled. The **no** form of this command disables debugging output.

[no] debug frame-relay foresight

Sample Display

Figure 2-77 shows the display message returned in response to the **debug frame-relay foresight** command.

Figure 2-77 Sample Debug Frame-Relay Foresight Output

```
Router# debug frame-relay foresight  
  
FR rate control for DLCI 17 due to ForeSight msg
```

This message indicates the router learned from the ForeSight message that DLCI 17 is now experiencing congestion. The output rate for this circuit should be slowed down, and in the router this DLCI is configured to adapt traffic shaping in response to foresight messages.

Related Command

show frame-relay pvc

debug frame-relay informationelements

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

[no] debug frame-relay informationelements

Usage Guidelines

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

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

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

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

Sample Display

Figure 2-78 shows sample **debug frame-relay informationelements** output. In this example, each information element has a length associated with it. For those with odd-numbered lengths, only the specified bytes are valid, and the extra byte is invalid. For example, in the message “Call Ref, length: 3, 0x0200 0x0100,” only “02 00 01” is valid, the last “00” is invalid.

Figure 2-78 Sample Debug Frame-Relay Information Elements Output

```
1w0d# debug frame-relay informationelements

1w0d: Outgoing MSG_SETUP

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

```
1w0d: Dir: U --> N, Type: Low Lyr Comp, length: 5, 0x7C03 0x88A0 0xCE65
1w0d: Dir: U --> N, Type: User to User, length: 4, 0x7E02 0x0000
```

Table 2-40 explains the information elements in the example shown in Figure 2-78.

Table 2-40 Information Elements in a Setup Message

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

Related Command

debug frame-relay callcontrol

debug frame-relay lapf

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

[no] debug frame-relay lapf

Usage Guidelines

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

Sample Displays

Figure 2-79 shows sample **debug frame-relay lapf** output. In this example, a line being brought up indicates an exchange of set asynchronous balanced mode extended (SAMBE) and unnumbered acknowledgment (UA) commands. A SABME is initiated by both sides, and a UA is the response. Until the SABME gets a UA response, the line is not declared to be up. The *p/f* value indicates the poll/final bit setting. *TX* means *send*, and *RX* means *receive*.

Figure 2-79 Sample Debug Frame-Relay LAPF Output—SABME-UA Exchange

```
1w0d# debug frame-relay lapf

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

In the example shown in Figure 2-80, a line in an up LAPF state should see a steady exchange of RR (receiver ready) messages. *TX* means *send*, *RX* means *receive*, and *N(R)* indicates the receive sequence number.

Figure 2-80 Sample Debug Frame-Relay LAPF Output—LAPF State

```
1w0d# debug frame-relay lapf

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

debug frame-relay lmi

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

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

Syntax Description

interface name (Optional) Name of interface.

Usage Guidelines

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

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

Sample Display

Figure 2-81 shows sample **debug frame-relay lmi** output.

Figure 2-81 Sample Debug Frame-Relay LMI Output

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

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

S2546

In Figure 2-81, the first four lines describe an LMI exchange. The first line describes the LMI request the router has sent to the switch. The second line describes the LMI reply the router has received from the switch. The third and fourth lines describe the response to this request from the switch. This

LMI exchange is followed by two similar LMI exchanges. The last six lines in Figure 2-81 consist of a full LMI status message that includes a description of the router’s two permanent virtual circuits (PVCs).

Table 2-41 describes significant fields in the first line of the **debug frame-relay lmi** output shown in Figure 2-81.

Table 2-41 Debug Frame-Relay LMI Field Descriptions—Part 1

Field	Description
Serial1(out)	Indication that the LMI request was sent out on the Serial1 interface.
StEnq	Command mode of message: StEnq—Status inquiry Status—Status reply
clock 20212760	System clock (in milliseconds). Useful for determining whether an appropriate amount of time has transpired between events.
myseq 206	The myseq counter maps to the router’s CURRENT SEQ counter.
yourseen 136	The yourseen counter maps to the LAST RCVD SEQ counter of the switch.
DTE up	Line protocol up/down state for the DTE (user) port.

Table 2-42 describes significant fields in the third and fourth lines of **debug frame-relay lmi** output shown in Figure 2-81.

Table 2-42 Debug Frame-Relay LMI Field Descriptions—Part 2

Field	Description
RT IE 1	Value of the report type information element.
length 1	Length of the report type information element (in bytes).
type 1	Report type in RT IE.
KA IE 3	Value of the keepalive information element.
length 2	Length of the keepalive information element (in bytes).
yourseq 138	The yourseq counter maps to the CURRENT SEQ counter of the switch.
myseq 206	The myseq counter maps to the router’s CURRENT SEQ counter.

Table 2-43 describes significant fields in the last line of **debug frame-relay lmi** output shown in Figure 2-81.

Table 2-43 Debug Frame-Relay LMI Field Descriptions—Part 3

Field	Description
PVC IE 0x7	Value of the permanent virtual circuit information element type.
length 0x6	Length of the PVC IE (in bytes).
dldci 401	DLCI decimal value for this PVC.

Table 2-43 Debug Frame-Relay LMI Field Descriptions—Part 3 (Continued)

Field	Description
status 0	Status value. Possible values include the following: 0x00—Added/inactive 0x02—Added/active 0x04—Deleted 0x08—New/inactive 0x0a—New/active
bw 56000	CIR (committed information rate), in decimal, for the DLCI.

debug frame-relay networklayerinterface

Use the **debug frame-relay networklayerinterface** EXEC command to display Network Layer Interface (NLI) information. The **no** form of this command disables debugging output.

[no] debug frame-relay networklayerinterface

Usage Guidelines

The Frame Relay SVC signaling subsystem is decoupled from the rest of the router code by means of the Network Layer Interface intermediate software layer.

The **debug frame-relay networklayerinterface** command shows what happens within the network layer interface when a call is set up or torn down. All output that contains an *NL* relate to the interaction between the Q.933 signaling subsystem and the Network Layer Interface.

Note The **debug frame-relay networklayerinterface** command has no significance to anyone who is not familiar with the inner workings of the Cisco IOS software. This command is typically used by service personnel to debug problem situations.

Sample Displays

Figure 2-82 shows sample **debug frame-relay networklayerinterface** output. This example displays the output generated when a call is set up. Figure 2-83 shows an example of the output generated when a call is torn down.

Figure 2-82 Sample Debug Frame-Relay Network Layer Interface Output—Call Setup

```
1w0d# debug frame-relay networklayerinterface

1w0d: NLI STATE: L3_CALL_REQ, Call ID 1 state 0
1w0d: NLI: Walking the event table 1
1w0d: NLI: Walking the event table 2
1w0d: NLI: Walking the event table 3
1w0d: NLI: Walking the event table 4
1w0d: NLI: Walking the event table 5
1w0d: NLI: Walking the event table 6
1w0d: NLI: Walking the event table 7
1w0d: NLI: Walking the event table 8
1w0d: NLI: Walking the event table 9
1w0d: NLI: NL0_L3CallReq
1w0d: NLI: State: STATE_NL_NULL, Event: L3_CALL_REQ, Next: STATE_L3_CALL_REQ
1w0d: NLI: Enqueued outgoing packet on holdq
1w0d: NLI: Map-list search: Found maplist bermuda
1w0d: daddr.subaddr 0, saddr.subaddr 0, saddr.subaddr 0
1w0d: saddr.subaddr 0, daddr.subaddr 0, daddr.subaddr 0
1w0d: nli_parameter_negotiation
1w0d: NLI STATE: NL_CALL_CNF, Call ID 1 state 10
1w0d: NLI: Walking the event table 1
1w0d: NLI: Walking the event table 2
1w0d: NLI: Walking the event table 3
1w0d: NLI: NLx_CallCnf
1w0d: NLI: State: STATE_L3_CALL_REQ, Event: NL_CALL_CNF, Next: STATE_NL_CALL_CNF
1w0d: Checking maplist "junk"
1w0d: working with maplist "bermuda"
1w0d: Checking maplist "bermuda"
```

```
1w0d: working with maplist "bermuda"
1w0d: NLI: Emptying holdQ, link 7, dlci 100, size 104
```

Figure 2-83 Sample Debug Frame-Relay Network Layer Interface Output—Call Teardown

```
1w0d# debug frame-relay networklayerinterface

1w0d: NLI: L3 Call Release Req for Call ID 1
1w0d: NLI STATE: L3_CALL_REL_REQ, Call ID 1 state 3
1w0d: NLI: Walking the event table 1
1w0d: NLI: Walking the event table 2
1w0d: NLI: Walking the event table 3
1w0d: NLI: Walking the event table 4
1w0d: NLI: Walking the event table 5
1w0d: NLI: Walking the event table 6
1w0d: NLI: Walking the event table 7
1w0d: NLI: Walking the event table 8
1w0d: NLI: Walking the event table 9
1w0d: NLI: Walking the event table 10
1w0d: NLI: NLx_L3CallRej
1w0d: NLI: State: STATE_NL_CALL_CNF, Event: L3_CALL_REL_REQ, Next: STATE_L3_CALL_REL_REQ
1w0d: NLI: junk: State: STATE_NL_NULL, Event: L3_CALL_REL_REQ, Next: STATE_NL_NULL
1w0d: NLI: Map-list search: Found maplist junk
1w0d: daddr.subaddr 0, saddr.subaddr 0, saddr.subaddr 0
1w0d: saddr.subaddr 0, daddr.subaddr 0, daddr.subaddr 0
1w0d: nli_parameter_negotiation
1w0d: NLI STATE: NL_REL_CNF, Call ID 1 state 0
1w0d: NLI: Walking the event table 1
1w0d: NLI: Walking the event table 2
1w0d: NLI: Walking the event table 3
1w0d: NLI: Walking the event table 4
1w0d: NLI: Walking the event table 5
1w0d: NLI: Walking the event table 6
1w0d: NLI: Walking the event table 7
1w0d: NLI: NLx_RelCnf
1w0d: NLI: State: STATE_NL_NULL, Event: NL_REL_CNF, Next: STATE_NL_NULL
```

Table 2-44 describes the states and events shown in Figure 2-82 and Figure 2-83.

Table 2-44 Network Layer Interface State and Event Descriptions

State and Event	Description
L3_CALL_REQ	Internal call setup request. Network layer indicates that a switched virtual circuit (SVC) is required.
STATE_NL_NULL	Call in initial state—no call exists.
STATE_L3_CALL_REQ	Setup message sent out and waiting for a reply. This is the state the network layer state machine transitions to when a call request is received from Layer 3 but no confirmation has been received from the network.
NL_CALL_CNF	Message sent from Q.933 signaling subsystem to the Network Layer Interface asking that internal resources be allocated for the call.
STATE_L3_CALL_CNF	Q.933 state indicating that the call is active. After the network confirms a call request using a connect message, the Q.933 state machine transitions to this state.
STATE_NL_CALL_CNF	Internal software state indicating software resources are assigned and the call is up. After Q.933 transitions to the STATE_L3_CALL_CNF state, it sends an NL_CALL_CNF message to the network layer state machine, which then transitions to the STATE_NL_CALL_CNF state.

Table 2-44 Network Layer Interface State and Event Descriptions (Continued)

State and Event	Description
L3_CALL_REL_REQ	Internal request to release the call.
STATE_L3_CALL_REL_REQ	Internal software state indicating the call is in the process of being released. At this point, the Q.933 subsystem is told that the call is being released and a disconnect message goes out for the Q.933 subsystem.
NL_REL_CNF	Indication from the Q.933 signaling subsystem that the signaling subsystem is releasing the call. After receiving a release complete message from the network indicating that the release process is complete, the Q.933 subsystem sends an NL_REL_CNF event to the network layer subsystem.

Related Command

debug frame-relay callcontrol

debug frame-relay packet

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

[no] debug frame-relay packet [interface name [dlci value]]

Syntax Description

interface name (Optional) Name of interface or subinterface.

dlci value (Optional) Data link connection identifier (DLCI) decimal value.

Usage Guidelines

This command helps you analyze the packets that are sent on a Frame Relay interface. Because the **debug frame-relay packet** command generates large amounts of output, only use it when traffic on the Frame Relay network is less than 25 packets per second. Use the options to limit the debugging output to a specific DLCI or interface.

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

Sample Display

Figure 2-84 shows sample **debug frame-relay packet** output.

Figure 2-84 Sample Debug Frame-Relay Packet Output

```
router# debug frame-relay packets
```

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

```
Serial0: broadcast = 0, link 809B, addr 10.2
Serial0(o):DLCI 100 type 809B size 104
```

```
Serial0: broadcast search
Serial0(o):DLCI 300 type 809B size 24
Serial0(o):DLCI 400 type 809B size 24
```

S2547

Groups of output lines

As Figure 2-84 shows, **debug frame-relay packet** output consists of groups of output lines; each group describes a Frame Relay packet that has been sent. The number of lines in the group can vary, depending on the number of data link connection identifiers (DLCIs) on which the packet was sent. For example, the first two pairs of output lines describe two different packets, both of which were sent out on a single DLCI. The last three lines in Figure 2-84 describe a single Frame Relay packet that was sent out on two DLCIs.

Table 2-45 describes significant fields shown in the first pair of output lines in Figure 2-84.

Table 2-45 Debug Frame-Relay Packet Field Descriptions

Field	Description
Serial0:	Interface that has sent the Frame Relay packet.
broadcast = 1	Destination of the packet. Possible values include the following: broadcast = 1—Broadcast address broadcast = 0—Particular destination broadcast search—Searches all Frame Relay map entries for this particular protocol that include the keyword broadcast .
link 809B	Link type, as documented under debug frame-relay .
addr 65535.255	Destination protocol address for this packet. In this case, it is an AppleTalk address.
Serial0(o):	(o) indicates that this is an output event.
DLCI 500	Decimal value of the DLCI.
type 809B	Packet type, as documented under debug frame-relay .
size 24	Size of this packet (in bytes).

Explanations for other lines of output shown in Figure 2-84 follow.

The following lines describe a Frame Relay packet sent to a particular address; in this case AppleTalk address 10.2:

```
Serial0: broadcast - 0, link 809B, addr 10.2
Serial0(o):DLCI 100 type 809B size 104
```

The following lines describe a Frame Relay packet that went out on two different DLCIs, because two Frame Relay map entries were found:

```
Serial0: broadcast search
Serial0(o):DLCI 300 type 809B size 24
Serial0(o):DLCI 400 type 809B size 24
```

The following lines do not appear in Figure 2-84. They describe a Frame Relay packet sent to a true broadcast address.

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

debug fras error

Use the **debug fras error** EXEC command to display information about Frame Relay Access Support (FRAS) protocol errors. The **no** form of this command disables debugging output.

[no] debug fras error

Usage Guidelines

For complete information on the FRAS process, use the **debug fras message** along with the **debug fras error** command.

Sample Display

Figure 2-85 shows sample **debug fras error** output. This example shows that no logical connection exists between the local station and remote station in the current setup.

Figure 2-85 Sample Debug FRAS Error Output

```
Router# debug fras error

FRAS: No route, lmac 1000.5acc.7fb1 rmac 4fff.0000.0000, lSap=0x4, rSap=0x4
FRAS: Can not find the Setup
```

Related Commands

debug cls message
debug fras message
debug fras state

debug fras-host activation

Use the **debug fras-host activation** EXEC command to display the LLC2 session activation and deactivation frames (such as XID, SABME, DISC, UA) that are being handled by FRAS Host. The **no** form of this command disables debugging output.

[no] debug fras-host activation

Usage Guidelines

If many LLC2 sessions are being activated or deactivated at any time, this command may generate large amounts of output to the console.

Sample Display

Figure 2-86 shows sample **debug fras-host activation** output.

Figure 2-86 Sample Debug FRAS-Host Activation Output

```
Router# debug fras-host activation
FRHOST: Snd      TST C to HOST, DA = 4001.3745.1088 SA = 400f.ddd.001e DSAP = 0x00 SSAP = 0x04
FRHOST: Fwd  BNN  XID to HOST, DA = 4001.3745.1088 SA = 400f.ddd.001e DSAP = 0x04 SSAP = 0x04
FRHOST: Fwd HOST  XID to  BNN, DA = 400f.ddd.001e SA = 4001.3745.1088 DSAP = 0x04 SSAP = 0x05
FRHOST: Fwd  BNN  XID to HOST, DA = 4001.3745.1088 SA = 400f.ddd.001e DSAP = 0x04 SSAP = 0x04
FRHOST: Fwd HOST SABME to BNN, DA = 400f.ddd.001e SA = 4001.3745.1088 DSAP = 0x04 SSAP = 0x04
FRHOST: Fwd  BNN  UA to HOST, DA = 4001.3745.1088 SA = 400f.ddd.001e DSAP = 0x04 SSAP = 0x05
```

The first line indicates that FRAS Host sent a TEST Command to the host. In the second line, the FRAS Host forwards an XID frame from a BNN device to the host. In the third line, FRAS Host forwards an XID from the host to the BNN device. Table 2-46 describes the common fields in these lines of output.

Table 2-46 Debug FRAS-Host Activation Field Descriptions

Field	Description
DA	Destination MAC address of the frame.
SA	Source MAC address of the frame.
DSAP	Destination SAP of the frame.
SSAP	Source SAP of the frame.

debug fras-host error

Use the **debug fras-host error** EXEC command to enable FRAS Host to send error messages to the console. The **no** form of this command disables debugging output.

[no] debug fras-host error

Sample Display

Figure 2-87 shows sample **debug fras-host error** output when the I-field in a TEST Response frame from a host does not match the I-field of the TEST Command sent by the FRAS Host.

Figure 2-87 Sample Debug FRAS-Host Error Output

```
Router# debug fras-host error  
  
FRHOST: SRB TST R Protocol Violation - LLC I-field not maintained.
```

debug fras-host packet

Use the **debug fras-host packet** EXEC command to see what LLC2 session frames are being handled by FRAS Host. The **no** form of this command disables debugging output.

[no] debug fras-host packet

Usage Guidelines

Use this command with great care. If many LLC2 sessions are active and passing data, this command may generate a tremendous amount of output to the console and impact performance.

Sample Display

Figure 2-88 shows sample **debug fras-host packet** output.

Figure 2-88 Sample Debug FRAS-Host Packet Output

```
Router# debug fras-host packet

FRHOST: Snd      TST C to HOST, DA = 4001.3745.1088 SA = 400f.ddd.001e DSAP = 0x00 SSAP = 0x04
FRHOST: Fwd  BNN  XID to HOST, DA = 4001.3745.1088 SA = 400f.ddd.001e DSAP = 0x04 SSAP = 0x04
FRHOST: Fwd  HOST  XID to BNN, DA = 400f.ddd.001e SA = 4001.3745.1088 DSAP = 0x04 SSAP = 0x05
FRHOST: Fwd  BNN  XID to HOST, DA = 4001.3745.1088 SA = 400f.ddd.001e DSAP = 0x04 SSAP = 0x04
FRHOST: Fwd  HOST  SABME to BNN, DA = 400f.ddd.001e SA = 4001.3745.1088 DSAP = 0x04 SSAP = 0x04
FRHOST: Fwd  BNN  UA to HOST, DA = 4001.3745.1088 SA = 400f.ddd.001e DSAP = 0x04 SSAP = 0x05
FRHOST: Fwd  HOST  LLC-2 to BNN, DA = 400f.ddd.001e SA = 4001.3745.1088 DSAP = 0x04 SSAP = 0x04
FRHOST: Fwd  BNN  LLC-2 to HOST, DA = 4001.3745.1088 SA = 400f.ddd.001e DSAP = 0x04 SSAP = 0x05
FRHOST: Fwd  HOST  LLC-2 to BNN, DA = 400f.ddd.001e SA = 4001.3745.1088 DSAP = 0x04 SSAP = 0x04
FRHOST: Fwd  BNN  LLC-2 to HOST, DA = 4001.3745.1088 SA = 400f.ddd.001e DSAP = 0x04 SSAP = 0x04
```

The **debug fras-host packet** output contains all of the output from the **debug fras-host activation** command as well as additional information. The first six lines of this sample display are the same as the output from the **debug fras-host activation** command. The last lines show LLC-2 frames being transmitted between the BNN device and the host. Table 2-47 describes the common fields in these lines of output.

Table 2-47 Debug FRAS-Host Packet Field Descriptions

Field	Description
DA	Destination MAC address of the frame.
SA	Source MAC address of the frame.
DSAP	Destination SAP of the frame.
SSAP	Source SAP of the frame.

debug fras-host snmp

Use the **debug fras-host snmp** EXEC command to display messages to the console describing SNMP requests to the FRAS Host MIB. The **no** form of this command disables debugging output.

[no] debug fras-host snmp

Usage Guidelines

Use of this command may result in large amounts of output to the screen. Only use this command for problem determination.

Sample Display

Figure 2-89 shows sample **debug fras-host snmp** output. In this example, the MIB variable `k_frasHostConnEntry_get()` is providing SNMP information for a FRAS host.

Figure 2-89 Sample Debug FRAS-Host SNMP Output

```
Router# debug fras-host snmp

k_frasHostConnEntry_get(): serNum = -1, vRingIfIdx = 31, frIfIdx = 12
Hmac = 4001.3745.1088, frLocSap = 4, Rmac = 400f.dddd.001e, frRemSap = 4
```

Table 2-48 describes fields shown in this sample output.

Table 2-48 Debug FRAS-Host SNMP Field Descriptions

Field	Description
serNum	Serial number of the SNMP request.
vRingIfIdx	Interface index of a virtual Token Ring.
frIfIdx	Interface index of a frame relay serial interface.
Hmac	MAC address associated with the host for this connection.
frLocSap	SAP associated with the host for this connection.
Rmac	MAC address associated with the FRAD for this connection.
frRemSap	LLC-2 SAP associated with the FRAD for this connection.

debug fras message

Use the **debug fras message** EXEC command to display general information about Frame Relay Access Support (FRAS) messages. The **no** form of this command disables debugging output.

[no] debug fras message

Usage Guidelines

For complete information on the FRAS process, use the **debug fras error** along with the **debug fras message** command.

Sample Display

Figure 2-90 shows sample **debug fras message** output. This example shows incoming Cisco Link Services (CLS) primitives.

Figure 2-90 Sample Debug FRAS Message Output

```
Router# debug fras message  
  
FRAS: receive 4C23  
FRAS: receive CC09
```

Related Commands

debug cls message
debug fras error
debug fras state

debug fras state

Use the **debug fras state** EXEC command to display information about Frame Relay Access Support (FRAS) data link control link state changes. The **no** form of this command disables debugging output.

[no] debug fras state

Sample Display

Figure 2-91 shows sample **debug fras state** output. This example shows the state changing from a *request open station is sent* state to an *exchange XID* state.

Possible states are the following: reset, request open station is sent, exchange xid, connection request is sent, signal station wait, connection response wait, connection response sent, connection established, disconnect wait, and number of link states.

Figure 2-91 Sample Debug FRAS State Output

```
Router# debug fras state

FRAS: TR0 (04/04) oldstate=LS_RQOPNSTNSENT, input=RQ_OPNSTN_CNF
FRAS: newstate=LS_EXCHGXID
```

Related Commands

debug cls message
debug fras error
debug fras message