

## 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

<b>border-peers</b>	(Optional) Enables debugging output for border peer events.
<b>interface</b> <i>interface</i>	(Optional) Specifies a remote peer to debug by a direct interface.
<b>ip address</b> <i>ip-address</i>	(Optional) Specifies a remote peer to debug by its IP address.
<b>core</b>	(Optional) Enables debugging output for DLSw core events.
<b>flow-control</b>	(Optional) Enables debugging output for congestion in the WAN or at the remote end station.
<b>messages</b>	(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.
<b>state</b>	(Optional) Enables debugging output for state changes on the circuit.
<b>xid</b>	(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.
<b>local-circuit</b> <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.
<b>peers</b>	(Optional) Enables debugging output for peer events.
<b>fast-errors</b>	(Optional) Debugs errors for fast-switched packets.
<b>fast-paks</b>	(Optional) Debugs fast-switched packets.
<b>fst-seq</b>	(Optional) Debug FST sequence numbers on fast switched packets.
<b>udp</b>	(Optional) Debug UDP packets.

<b>reachability</b>	(Optional) Enables debugging output for reachability events (explorer traffic). If no options are specified, event-level information is displayed for all protocols.
<b>error   verbose</b>	(Optional) Specifies how much reachability information you want displayed. The <b>verbose</b> keyword displays everything, including errors and events. The <b>error</b> keyword displays error information only. If no option is specified, event-level information is displayed.
<b>sna   netbios</b>	(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 Display

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 xchged 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 xchged 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-FSTPFISM (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-FSTPFISM (peer 10.10.10.4(0)): state:DISCONN->WAIT_CAP
DLSw: START-FSTPFISM (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-FSTPFISM (peer 10.10.10.4(0)): state:WAIT_CAP->WAIT_CAP
DLSw: START-FSTPFISM (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-FSTPFISM (peer 10.10.10.4(0)): state:WAIT_CAP->WAIT_CAP
DLSw: Processing delayed event:SSP-CAP EXCHANGED - prev state:WAIT_CAP
DLSw: START-FSTPFISM (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-FSTPFISM (peer 10.10.10.4(0)): state:WAIT_CAP->CONNECT
```

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

```
DLSw: START-FSTPFISM (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-FSTPFISM (peer 10.3.8.7(0)): state:DISCONN->WAIT_CAP
DLSw: START-FSTPFISM (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-FSTPFISM (peer 10.3.8.7(0)): state:WAIT_CAP->WAIT_CAP
DLSw: Processing delayed event:SSP-CAP EXCHANGED - prev state:WAIT_CAP
DLSw: START-FSTPFISM (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-FSTPFISM (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.Reg  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.Reg  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-LLC2PFSM (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.Reg  dlen: 106
DLSw: END-LLC2PFSM (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.Req  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.Req  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-LLC2PFISM (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.Req  dlen: 106
DLSw: END-LLC2PFISM (peer on interface Serial0): state:DISCONN->CONS_PEND

DLSw: Peer Received : CLSI Msg : REQ_OPNSTN.Cfm CLS_OK dlen: 106
DLSw: START-LLC2PFISM (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.Req  dlen: 418
DLSw: CapExId Msg sent to peer on interface Serial0
DLSw: END-LLC2PFISM (peer on interface Serial0): state:CONS_PEND->WAIT_CAP

DLSw: Peer Received : CLSI Msg : CONNECTED.Ind  dlen: 8
DLSw: START-LLC2PFISM (peer on interface Serial0): event:CLS-CONNECTED.IND
state:WAIT_CAP
DLSw: END-LLC2PFISM (peer on interface Serial0): state:WAIT_CAP->WAIT_CAP

DLSw: Peer Received : CLSI Msg : DATA.Ind  dlen: 418
DLSw: START-LLC2PFISM (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-LLC2PFISM (peer on interface Serial0): state:WAIT_CAP->WAIT_CAP

DLSw: Peer Received : CLSI Msg : DATA.Ind  dlen: 96
DLSw: START-LLC2PFISM (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-LLC2PFISM (peer on interface Serial0): state:WAIT_CAP->WAIT_CAP

DLSw: Processing delayed event:SSP-CAP EXCHANGED - prev state:WAIT_CAP
DLSw: START-LLC2PFISM (peer on interface Serial0): event:SSP-CAP EXCHANGED
state:WAIT_CAP
DLSw: dllc2p_action_l() cap xchged for peer on interface Serial0
DLSw: END-LLC2PFISM (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 32 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 32** 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 drip event

Use the **debug drip event** privileged EXEC command to display debug messages for DRiP events. Use the **no** form of this command to disable debugging output.

**[no] debug drip event**

### Default

Debugging is disabled for DRiP events.

### Usage Guidelines

When a Fast Ethernet subinterface is configured for TRISL encapsulation and a TrCRF is defined, the DRiP protocol is activated. The DRiP protocol adds the VLAN ID specified in the router command to its database and recognizes the VLAN as a locally configured, active VLAN.

### Sample Display

The following is sample output from the **debug drip event** command:

```
75-2(config-subif)#encapsulation tr-isl trbrf-vlan 999 bridge-num 9
```

DRiP is initiated when a local VLAN is added to the DRiP database:

```
DRIP : init
```

The VLAN ID is added locally when TRISL is configured:

```
DRIP : configure vlanNo = 100  
DRIP is configuring the VLAN:
```

VLAN 100 is activated in the database:

```
DRIP : local status active for vlanNo = 100  
DRIP : resolve local - DRIP_VLAN_ACTIVE
```

DRiP acknowledges that a VLAN is active and is now capable of printing any debug information, if necessary:

```
DRIP Change notification active vlan 100  
DRIP : State notification  
DRIP Change notification active vlan 100
```

DRiP logs the new VLAN ID:

```
DRIP : configure - ADD_ID 2
```

DRiP will send an advertisement on all its trunk ports:

```
DRIP : configure - send_adv = TRUE
```

DRiP provides information of the trunk port and the length of the packet:

```
DRIP : transmit on 0000.0c50.1900, length = 24
```

DRiP gets a packet from the network:

```
612B92C0: 01000C00 00000000 0C501900 0000AAAA .....P....**
612B92D0: 0300000C 00020000 00000100 0CCCCCCC .....LLL
612B92E0: 00000C50 19000020 AAAA0300 000C0102 ...P... **.....
612B92F0: 01010114 00000002 00000002 00000C50 .....P
612B9300: 19000001 04C00064 04 .....@.d.
```

DRiP gets a packet from the network:

```
Recvd. pak
```

DRiP recognizes that the VLAN ID it is getting is a new one from the network:

```
6116C840:                0100 0CCCCCCC          ...LLL
6116C850: 00102F72 CFBF0024 AAAA0300 000C0102 ../rK{.$**.....
6116C860: 01FF0214 0002E254 00015003 00102F72 .....bT..P.../r
6116C870: C8000010 04C00014 044003EB 14      H....@...@.k.
DRIP : remote update - Never heard of this vlan
```

DRiP attempts to resolve any conflicts when it hears of a new VLAN. The value action = 1 means to notify the local platform of change in state:

```
DRIP : resolve remote for vlan 20 in FastEthernet0/0/0
DRIP : resolve remote - action = 1
```

The local platform is notified of change in state:

```
DRIP Change notification active vlan 20
```

Another new VLAN ID was received in the packet:

```
DRIP : resolve remote for vlan 1003 in FastEthernet0/0/0
```

No action is required:

```
DRIP : resolve remote - action = 0
```

Thirty seconds have expired, and DRiP sends its local database entries to all its trunk ports:

```
DRIP : local timer expired
DRIP : transmit on 0000.0c50.1900, length = 24
612B92C0: 01000C00 00000000 0C501900 0000AAAA .....P....**
612B92D0: 0300000C 00020000 00000100 0CCCCCCC .....LLL
612B92E0: 00000C50 19000020 AAAA0300 000C0102 ...P... **.....
612B92F0: 01FF0114 00000003 00000002 00000C50 .....P
612B9300: 19000001 04C00064 04 .....@.d.
```

## debug drip packet

Use the **debug drip packet** privileged EXEC command to display debug messages for DRiP packets. Use the **no** form of this command to disable debugging output.

**[no] debug drip packet**

### Default

Debugging is not enabled for DRIP packets.

### Usage Guidelines

Before you use this command, you can optionally use the **clear drip** command first. As a result the DRiP counters are reset to 0. If the drip counters begin to increment, the router is receiving packets.

### Sample Displays

The following is sample output from the **debug drip packet** command.

The following type of output is displayed when a packet is entering the router and you use the **show debug** command:

```
039E5FC0:      0100 0CCCCCCC 00E0A39B 3FFB0028      ...LLL.~#.?{.(
039E5FD0: AAAA0300 000C0102 01FF0314 0000A5F6  **.....%v
039E5FE0: 00008805 00E0A39B 3C000000 04C00028      ....~#.<....@.(
039E5FF0: 04C00032 044003EB 0F                .@.2.@.k.
039FBD20:                01000C00 00000010      .....
```

The following type of output is displayed when a packet is transmitted by the router:

```
039FBD30: A6AEB450 0000AAAA 0300000C 00020000  &.4P.**.....
039FBD40: 00000100 0CCCCCCC 0010A6AE B4500020  ....LLL..&.4P.
039FBD50: AAAA0300 000C0102 01FF0114 00000003  **.....
039FBD60: 00000002 0010A6AE B4500001 04C00064  ....&.4P...@.d
039FBD70: 04                .
```

### Related Commands

**debug drip event**  
**encapsulation tri-isl**

## 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

The following is sample output from the **debug dspu activation** command. Not all intermediate numbers are shown for the “activated” and “deactivated” logical unit (LU) address ranges.

```
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 33 describes significant fields in the output.

**Table 33** 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.
HOST3745-253	Host name or PU name and the LU address, separated by a dash.

**Table 33**      **Debug DSPU Activation Field Descriptions (Continued)**

<b>Field</b>	<b>Description</b>
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

The following is sample output from the **debug dspu packet** command:

```
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 34 describes significant fields in the output.

**Table 34** 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 drip event**  
**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

The following is sample output from the **debug dspu state** command. Not all intermediate numbers are shown for the “activated” and “deactivated” logical unit (LU) address ranges.

```
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 35 describes significant fields in the output.

**Table 35** Debug DSPU State 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.

**Table 35**      **Debug DSPU State Field Descriptions (Continued)**

<b>Field</b>	<b>Description</b>
HOST3745-253	Host name or PU name and LU address.
<i>input=input,</i>	The input received by the FSM.
<i>previous-state, -&gt; current-state</i>	The previous state and current new state as seen by the FSM.

**Related Commands**

**debug drip event**  
**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

The following is sample output from the **debug dspu trace** command:

```
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 36 describes significant fields in the output.

**Table 36** 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.
LU	A logical unit (LU) event triggered the message.
HOST3745-253	Host name or PU name and LU address.

**Table 36** Debug DSPU Trace Field Descriptions (Continued)

<b>Field</b>	<b>Description</b>
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 drip event**  
**debug dspu packet**  
**debug dspu state**

## debug eigrp fsm

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

**[no] debug eigrp fsm**

### Usage Guidelines

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

### Sample Display

The following is sample output from the **debug eigrp fsm** command:

```
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 4294967295 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
```

In the first line, DUAL stands for Diffusing Update ALgorithm. It is the basic mechanism within EIGRP 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 hold-down.

In the following output, EIGRP 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

The following is sample output from the **debug eigrp packet** command:

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

Table 37 describes significant fields in the output.

**Table 37** Debug EIGRP Packet Field Descriptions

Field	Description
EIGRP:	EIGRP packet information.
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 37      Debug EIGRP Packet Field Descriptions (Continued)**

<b>Field</b>	<b>Description</b>
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

The following is sample output from the **debug fddi smt-packets** command. 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.

```
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 38 describes the fields in the output.

**Table 38** 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

The following is sample output from the **debug frame-relay** command:

```
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 39 describes significant fields.

**Table 39** 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 39**      **Debug Frame-Relay Field Descriptions (Continued)**

<b>Field</b>	<b>Description</b>
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 40 explains the possible call states.

**Table 40** 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.
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

The following is sample output from the **debug frame-relay events** command:

```
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 the output 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

The following is sample output from the shows the display message returned in response to the **debug frame-relay foresight** command.

```
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

The following is sample output from the **debug frame-relay informationelements** command. 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.

```
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 41 explains the information elements in the example shown.

**Table 41** 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 ip tcp header-compression

To display debugging information about TCP/IP header compression on Frame Relay interfaces, use the **debug frame-relay ip tcp header-compression** command in privileged EXEC mode. To disable debugging output, use the **no** form of this command.

**[no] debug frame-relay ip tcp header-compression**

### Usage Guidelines

The **debug frame-relay ip tcp header-compression** command shows the control packets that are passed to initialize IP header compression (IPHC) on a permanent virtual circuit (PVC). For Cisco IPHC, typically two packets are passed: one sent and one received per PVC. (Inverse Address Resolution Protocol (InARP) packets are sent on PVCs that do not have a mapping defined between a destination protocol address and the data-link connection identifier (DLCI) or Frame Relay PVC bundle that connects to the destination address.)

Debug messages are displayed only if the IPHC control protocol is renegotiated (for an interface or PVC state change or for a configuration change).

### Sample Display

The following is sample output from the **debug frame-relay ip tcp header-compression** command when Cisco IPHC is configured in the IPHC profile:

```
Router# debug frame-relay ip tcp header-compression

*Nov 14 09:22:07.991: InARP REQ: Tx compr_flags 43 *Nov 14 09:22:08.103: InARP RSP: Rx
compr_flags: 43
```

Table 42 describes the significant fields shown in the display.

**Table 42** Debug Frame Relay IP TCP Header Compression Field Descriptions

Field	Description
InARP REQ: Tx	Indicates that an InARP request was sent or received. Following are the possible values: <ul style="list-style-type: none"> <li>InARP REQ Tx—An InARP request was sent.</li> <li>InARP REQ Rx—An InARP request was received.</li> </ul>
InARP RSP: Rx	Indicates that an InARP response was sent or received. Following are the possible values: <ul style="list-style-type: none"> <li>InARP REQ Tx—An InARP response was sent.</li> <li>InARP REQ Rx—An InARP response was received.</li> </ul>
compr_flags: 43	Compression flags that Frame Relay peers use to negotiate Cisco IPHC options. It consists of a bit mask, and the number is displayed in hexadecimal format. Following are the bits: <ul style="list-style-type: none"> <li>0x0001—TCP IPHC</li> <li>0x0002—RTP IPHC</li> <li>0x0004—Passive TCP compression</li> <li>0x0008—Passive RTP compression</li> <li>0x0040—Frame Relay IPHC options</li> </ul>

## 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 Display

The following is sample output from the **debug frame-relay lapf** command. In this example, a line being brought up indicates an exchange of set asynchronous balanced mode extended (SABME) 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.

```
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 following example, 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.

```
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

The following is sample output from the **debug frame-relay lmi** command:

```

router# debug frame-relay lmi

LMI exchange — Serial1(out): StEnq, clock 20212760, myseq 206, mineseen 205, yourseen 136, DTE up
                  Serial1(in): Status, clock 20212764, myseq 206
                  RT IE 1, length 1, type 1
                  KA IE 3, length 2, yourseq 138, myseq 206
                  Serial1(out): StEnq, clock 20222760, myseq 207, mineseen 206, yourseen 138, DTE up
                  Serial1(in): Status, clock 20222764, myseq 207
                  RT IE 1, length 1, type 1
                  KA IE 3, length 2, yourseq 140, myseq 207
                  Serial1(out): clock 20232760, myseq 208, mineseen 207, yourseen 140, line up
                  RT IE 1, length 1, type 1
                  KA IE 3, length 2, yourseq 142, myseq 208
Full LMI status message — 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

```

S2546

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 consist of a full LMI status message that includes a description of the router's two permanent virtual circuits (PVCs).

Table 43 describes significant fields in the first line of the **debug frame-relay lmi** output.

**Table 43** 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.
yourseq 136	The yourseq counter maps to the LAST RCVD SEQ counter of the switch.
DTE up	Line protocol up/down state for the DTE (user) port.

Table 44 describes significant fields in the third and fourth lines of **debug frame-relay lmi** output.

**Table 44** 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 45 describes significant fields in the last line of **debug frame-relay lmi** output.

**Table 45      Debug Frame-Relay LMI Field Descriptions—Part 3**

<b>Field</b>	<b>Description</b>
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.
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

The following is sample output from the **debug frame-relay networklayerinterface** command. This example displays the output generated when a call is set up. The second example shows the output generated when a call is torn down.

```
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
```

```

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 46 describes the states and events in the output.

**Table 46 Network Layer Interface State and Event Descriptions**

State and Event	Description
L3_CALL_REQ	Internal call setup request. Network layer indicates that a switched virtual circuit (SVC) is required.
STATE_NL_NULL	Call in initial state—no call exists.
STATE_L3_CALL_REQ	Setup message sent out and waiting for a reply. This is the state the network layer state machine transitions to when a call request is received from Layer 3 but no confirmation has been received from the network.
NL_CALL_CNF	Message sent from Q.933 signaling subsystem to the Network Layer Interface asking that internal resources be allocated for the call.
STATE_L3_CALL_CNF	Q.933 state indicating that the call is active. After the network confirms a call request using a connect message, the Q.933 state machine transitions to this state.
STATE_NL_CALL_CNF	Internal software state indicating software resources are assigned and the call is up. After Q.933 transitions to the STATE_L3_CALL_CNF state, it sends an NL_CALL_CNF message to the network layer state machine, which then transitions to the STATE_NL_CALL_CNF state.
L3_CALL_REL_REQ	Internal request to release the call.

**Table 46 Network Layer Interface State and Event Descriptions (Continued)**

<b>State and Event</b>	<b>Description</b>
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

The following is sample output from the **debug frame-relay packet** command:

```

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
    
```

Groups of output lines

S2547

The **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 describe a single Frame Relay packet that was sent out on two DLCIs.

Table 47 describes significant fields shown in the first pair of output lines.

**Table 47** 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 <b>broadcast</b> .
link 809B	Link type, as documented under <b>debug frame-relay</b> .
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 <b>debug frame-relay</b> .
size 24	Size of this packet (in bytes).

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. They describe a Frame Relay packet sent to a true broadcast address.

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

## debug frame-relay ppp

Use the **debug frame-relay ppp** EXEC command to display debugging information. To disable debugging output, use the **no** form of this command.

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

### Usage Guidelines

Displays error messages for link states and LMI status changes for PPP over Frame Relay sessions.

To debug process-switched packets, use the **debug frame-relay packet** and/or **debug ppp packet** commands. To analyze the packets that have been *sent* on a Frame Relay interface, use the **debug frame-relay packet** command.

The **debug frame-relay ppp** command is generated from process level switching only and is not CPU intensive.

### Sample Displays

The following is sample output from the **debug frame-relay ppp** command where the encapsulation failed for VC 100:

```
Router# debug frame-relay ppp
FR-PPP: encaps failed for FR VC 100 on Serial0 down
FR-PPP: input- Serial0 vc or va down, pak dropped
```

The following is sample output from the **debug frame-relay ppp** and **debug frame-relay packet** commands. This example shows a virtual interface (virtual interface 1) successfully establishing a PPP connection over PPP.

```
Router# debug frame-relay ppp
Router# debug frame-relay packet

Vil LCP: O CONFREQ [Closed] id 1 len 10
Vil LCP:   MagicNumber 0xE0638565 (0x0506E0638565)
Serial2/1(o): dlci 201(0x3091), NLPID 0x3CF(PPP), datagramsize 16
Vil PPP: I pkt type 0xC021, datagramsize 14
Vil LCP: I CONFACK [REQsent] id 1 len 10
Vil LCP:   MagicNumber 0xE0638565 (0x0506E0638565)
Vil PPP: I pkt type 0xC021, datagramsize 14
Vil LCP: I CONFREQ [ACKrcvd] id 6 len 10
Vil LCP:   MagicNumber 0x000EAD99 (0x0506000EAD99)
Vil LCP: O CONFACK [ACKrcvd] id 6 len 10
Vil LCP:   MagicNumber 0x000EAD99 (0x0506000EAD99)
Serial2/1(o): dlci 201(0x3091), NLPID 0x3CF(PPP), datagramsize 16
Vil IPCP: O CONFREQ [Closed] id 1 len 10
Vil IPCP:   Address 170.100.9.10 (0x0306AA64090A)
Serial2/1(o): dlci 201(0x3091), NLPID 0x3CF(PPP), datagramsize 16
Vil PPP: I pkt type 0x8021, datagramsize 14
Vil IPCP: I CONFREQ [REQsent] id 1 len 10
Vil IPCP:   Address 170.100.9.20 (0x0306AA640914)
Vil IPCP: O CONFACK [REQsent] id 1 len 10
Vil IPCP:   Address 170.100.9.20 (0x0306AA640914)
Serial2/1(o): dlci 201(0x3091), NLPID 0x3CF(PPP), datagramsize 16
Vil PPP: I pkt type 0x8021, datagramsize 14
Vil IPCP: I CONFACK [ACKsent] id 1 len 10
Vil IPCP:   Address 170.100.9.10 (0x0306AA64090A)
Vil PPP: I pkt type 0xC021, datagramsize 16
Vil LCP: I ECHOREQ [Open] id 1 len 12 magic 0x000EAD99
Vil LCP: O ECHOREP [Open] id 1 len 12 magic 0xE0638565
Serial2/1(o): dlci 201(0x3091), NLPID 0x3CF(PPP), datagramsize 18
```

```
Vil LCP: O ECHOREQ [Open] id 1 len 12 magic 0xE0638565
Serial2/1(o): dlci 201(0x3091), NLPID 0x3CF(PPP), datagramsize 18
Vil LCP: echo_cnt 4, sent id 1, line up
```

The following is sample output from the **debug frame-relay ppp** and **debug frame-relay packet** commands, which report a failed PPP over Frame Relay session. The problem is due to a challenge handshake authentication protocol (CHAP) failure.

```
Router# debug frame-relay ppp
Router# debug frame-relay packet

Vil LCP: O CONFREQ [Listen] id 24 len 10
Vil LCP:   MagicNumber 0xE068EC78 (0x0506E068EC78)
Serial2/1(o): dlci 201(0x3091), NLPID 0x3CF(PPP), datagramsize 16
Vil PPP: I pkt type 0xC021, datagramsize 19
Vil LCP: I CONFREQ [REQsent] id 18 len 15
Vil LCP:   AuthProto CHAP (0x0305C22305)
Vil LCP:   MagicNumber 0x0014387E (0x05060014387E)
Vil LCP: O CONFACK [REQsent] id 18 len 15
Vil LCP:   AuthProto CHAP (0x0305C22305)
Vil LCP:   MagicNumber 0x0014387E (0x05060014387E)
Serial2/1(o): dlci 201(0x3091), NLPID 0x3CF(PPP), datagramsize 21
Vil PPP: I pkt type 0xC021, datagramsize 14
Vil LCP: I CONFACK [ACKsent] id 24 len 10
Vil LCP:   MagicNumber 0xE068EC78 (0x0506E068EC78)
Vil PPP: I pkt type 0xC223, datagramsize 32
Vil CHAP: I CHALLENGE id 12 len 28 from "krishna"
Vil LCP: O TERMREQ [Open] id 25 len 4
Serial2/1(o): dlci 201(0x3091), NLPID 0x3CF(PPP), datagramsize 10
Vil PPP: I pkt type 0xC021, datagramsize 8
Vil LCP: I TERMACK [TERMsent] id 25 len 4
Serial2/1(i): dlci 201(0x3091), pkt type 0x2000, datagramsize 303
%SYS-5-CONFIG_I: Configured from console by console
Vil LCP: TIMEout: Time 0x199580 State Listen
```

## 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

The following is sample output from the **debug fras error** command. This example shows that no logical connection exists between the local station and remote station in the current setup.

```
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

The following is sample output from the **debug fras-host activation** command.

```
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 48 describes the common fields in these lines of output.

**Table 48** 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

The following is sample output from the **debug fras-host error** command 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.

```
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

The following is sample output from the **debug fras-host packet** command:

```
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 49 describes the common fields in these lines of output.

**Table 49** 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

The following is sample output from the **debug fras-host snmp** command. In this example, the MIB variable `k_frasHostConnEntry_get()` is providing SNMP information for a FRAS host.

```
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 50 describes fields shown in this sample output.

**Table 50**      **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

The following is sample output from the **debug fras message** command. This example shows incoming Cisco Link Services (CLS) primitives.

```
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

The following is sample output from the **debug fras state** command. 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.

```
Router# debug fras state

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

### Related Commands

**debug cls message**  
**debug fras error**  
**debug fras message**

## debug ftpserver

Use the **debug ftpserver EXEC** command to display information about the FTP server process. The **no** form of this command disables debugging output.

**[no] debug ftpserver**

### Sample Display

The following is sample output from the **debug ftpserver** command:

```
Router# debug ftpserver

Mar  3 10:21:10: %FTPSERVER-6-NEWCONN: FTP Server - new connection made.
-Process= "TCP/FTP Server", ipl= 0, pid= 53
Mar  3 10:21:10: FTPSRV_DEBUG:FTP Server file path: 'disk0:'
Mar  3 10:21:10: FTPSRV_DEBUG:(REPLY)  220
Mar  3 10:21:10: FTPSRV_DEBUG:FTProuter IOS-FTP server (version 1.00) ready.
Mar  3 10:21:10: FTPSRV_DEBUG:FTP Server Command received: 'USER aa'
Mar  3 10:21:20: FTPSRV_DEBUG:(REPLY)  331
Mar  3 10:21:20: FTPSRV_DEBUG>Password required for 'aa'.
Mar  3 10:21:20: FTPSRV_DEBUG:FTP Server Command received: 'PASS aa'
Mar  3 10:21:21: FTPSRV_DEBUG:(REPLY)  230
Mar  3 10:21:21: FTPSRV_DEBUG:Logged in.
Mar  3 10:21:21: FTPSRV_DEBUG:FTP Server Command received: 'SYST'
Mar  3 10:21:21: FTPSRV_DEBUG:(REPLY)  215
Mar  3 10:21:21: FTPSRV_DEBUG:Cisco IOS Type: L8 Version: IOS/FTP 1.00
Mar  3 10:21:21: FTPSRV_DEBUG:FTP Server Command received: 'PWD'
Mar  3 10:21:35: FTPSRV_DEBUG:(REPLY)  257
Mar  3 10:21:39: FTPSRV_DEBUG:FTP Server Command received: 'CWD disk0:/syslogd.d'r/'
Mar  3 10:21:45: FTPSRV_DEBUG:FTP Server file path: 'disk0:/syslogd.dir'
Mar  3 10:21:45: FTPSRV_DEBUG:(REPLY)  250
Mar  3 10:21:45: FTPSRV_DEBUG:CWD command successful.
Mar  3 10:21:45: FTPSRV_DEBUG:FTP Server Command received: 'PORT 171,69,30,20,22',32
Mar  3 10:21:46: FTPSRV_DEBUG:(REPLY)  200
Mar  3 10:21:46: FTPSRV_DEBUG:PORT command successful.
Mar  3 10:21:46: FTPSRV_DEBUG:FTP Server Command received: 'LIST'
Mar  3 10:21:47: FTPSRV_DEBUG:FTP Server file path: 'disk0:/syslogd.dir/.'
Mar  3 10:21:47: FTPSRV_DEBUG:(REPLY)  220
Mar  3 10:23:11: FTPSRV_DEBUG:Opening ASCII mode data connection for file list.
Mar  3 10:23:11: FTPSRV_DEBUG:(REPLY)  226
Mar  3 10:23:12: FTPSRV_DEBUG:Transfer complete.
Mar  3 10:23:12: FTPSRV_DEBUG:FTP Server Command received: 'TYPE I'
Mar  3 10:23:14: FTPSRV_DEBUG:(REPLY)  200
Mar  3 10:23:14: FTPSRV_DEBUG>Type set to I.
Mar  3 10:23:14: FTPSRV_DEBUG:FTP Server Command received: 'PORT 171,69,30,20,22',51
Mar  3 10:23:20: FTPSRV_DEBUG:(REPLY)  200
Mar  3 10:23:20: FTPSRV_DEBUG:PORT command successful.
Mar  3 10:23:20: FTPSRV_DEBUG:FTP Server Command received: 'RETR syslogd.1'
Mar  3 10:23:21: FTPSRV_DEBUG:FTP Server file path: 'disk0:/syslogd.dir/syslogd.1'
Mar  3 10:23:21: FTPSRV_DEBUG:FTPSERVER: Input path passed Top-dir(disk0:/syslogd.dir/)
test.
Mar  3 10:23:21: FTPSRV_DEBUG:(REPLY)  150
Mar  3 10:23:21: FTPSRV_DEBUG:Opening BINARY mode data connection for syslogd.1 (607317
bytes).
Mar  3 10:23:21: FTPSRV_DEBUG:(REPLY)  226
Mar  3 10:23:29: FTPSRV_DEBUG:Transfer complete.
```

The sample output corresponds to the following FTP client session. In this example, the user connects to the FTP server, views the contents of the top-level directory, and gets a file.

```
FTPclient% ftp FTProuter
Connected to FTProuter.cisco.com.
220 FTProuter IOS-FTP server (version 1.00) ready.
Name (FTProuter:me): aa
331 Password required for 'aa'.
Password:
230 Logged in.
Remote system type is Cisco.
ftp> pwd
257 "disk0:/syslogd.dir/" is current directory.
ftp> dir
200 PORT command successful.
150 Opening ASCII mode data connection for file list.
syslogd.1
syslogd.2
syslogd.3
syslogd.4
syslogd.5
syslogd.6
syslogd.7
syslogd.8
syslogd.9
syslogd.cur
226 Transfer complete.
ftp> bin
200 Type set to I.
ftp> get syslogd.1
200 PORT command successful.
150 Opening BINARY mode data connection for syslogd.1 (607317 bytes).
226 Transfer complete.
607317 bytes received in 7.7 seconds (77 Kbytes/s)
ftp>
```

The following **debug ftpserver** command output indicates that no top-level directory is specified. Therefore, the client cannot access any location on the FTP server. Use the **ftp-server topdir** command to specify the top-level directory.

```
Mar  3 10:29:14: FTPSRV_DEBUG:(REPLY)  550
Mar  3 10:29:14: FTPSRV_DEBUG:Access denied to 'disk0:'
```

## debug ip bgp

To display information related to processing BGPs, use the **debug ip bgp** privileged EXEC command. To disable the display of BGP information, use the **no** form of this command.

**debug ip bgp** [*A.B.C.D.* | **dampening** | **events** | **in** | **keepalives** | **out** | **updates** | **vpn4**]

**no debug ip bgp** [*A.B.C.D.* | **dampening** | **events** | **in** | **keepalives** | **out** | **updates** | **vpn4**]

### Syntax Description

<i>A.B.C.D.</i>	(Optional) Displays the BGP neighbor IP address.
<b>dampening</b>	(Optional) Displays BGP dampening.
<b>events</b>	(Optional) Displays BGP events.
<b>in</b>	(Optional) BGP inbound information.
<b>keepalives</b>	(Optional) Displays BGP keepalives.
<b>out</b>	(Optional) Displays BGP outbound information.
<b>updates</b>	(Optional) Displays BGP updates.
<b>vpn4</b>	(Optional) Displays VPNv4 NLRI information.

### Sample Display

The following is sample output from the **debug ip bgp** command:

```
Router# debug ip bgp vpn4

03:47:14:vpn:bgp_vpn4_bnetinit:100:2:58.0.0.0/8
03:47:14:vpn:bnettable add:100:2:58.0.0.0 / 8
03:47:14:vpn:bestpath_hook route_tag_change for vpn2:58.0.0.0/255.0.0.0(ok)
03:47:14:vpn:bgp_vpn4_bnetinit:100:2:57.0.0.0/8
03:47:14:vpn:bnettable add:100:2:57.0.0.0 / 8
03:47:14:vpn:bestpath_hook route_tag_change for vpn2:57.0.0.0/255.0.0.0(ok)
03:47:14:vpn:bgp_vpn4_bnetinit:100:2:14.0.0.0/8
03:47:14:vpn:bnettable add:100:2:14.0.0.0 / 8
03:47:14:vpn:bestpath_hook route_tag_chacle ip bgp *nge for vpn2:14.0.0.0/255.0.0.0(ok)
```

## debug ip cef

Use the **debug ip cef** EXEC commands for troubleshooting CEF.

```
debug ip cef { drops [access-list] | receive [access-list] | events [access-list] | prefix-ipc [access-list] | table [access-list] }
```

and

```
debug ip cef { ipc | interface-ipc }
```

### Syntax Description

<b>drops</b>	Records dropped packets.
<i>access-list</i>	(Optional) Controls collection of debugging information from specified lists
<b>receive</b>	Records packets that are not switched using information from the FIB table, but are received and sent to the next switching layer
<b>events</b>	Records general CEF events.
<b>prefix-ipc</b>	Records updates related to IP prefix information. Possible updates include: <ul style="list-style-type: none"><li>• Debugging of IP routing updates in a line card</li><li>• Reloading of a line card with a new table</li><li>• Route update from the route processor to the line card has exceeded the maximum number of routes</li><li>• Control messages related to FIB table prefixes</li></ul>
<b>table</b>	Produces a table showing events related to the FIB table. Possible types of events include: <ul style="list-style-type: none"><li>• Routing updates that populate the FIB table</li><li>• Flushing of the FIB table</li><li>• Adding or removing of entries to the FIB table</li><li>• Table reloading process</li></ul>
<b>ipc</b>	Records information related to IPC in CEF. Possible types of events include: <ul style="list-style-type: none"><li>• Transmission status of IPC messages</li><li>• Status of buffer space for ipc messages</li><li>• IPC messages received out of sequence</li><li>• Status of resequenced messages</li><li>• Throttle requests sent from a line card to the route processor</li></ul>
<b>interface-ipc</b>	Records IPC updates related to interfaces. Possible reporting includes an interface coming up or going down, updates to fibhwidb, fibidb, and so forth.

## debug ip cef accounting non-recursive

Use the **debug ip cef accounting non-recursive** EXEC commands to troubleshoot CEF accounting records. The **no** form of this command disables debugging output.

**[no] debug ip cef accounting non-recursive**

### Usage Guidelines

This command records accounting events for nonrecursive prefixes when the **ip cef accounting non-recursive** command is enabled in global configuration mode.

### Sample Display

The following is sample output from the **debug ip cef accounting non-recursive** command.

```
Router# debug ip cef accounting non-recursive

03:50:19:CEF-Acct:tmstats_binary:Beginning generation of tmstats
ephemeral file (mode binary)
03:50:19:CEF-Acct:snaphoting loadinfo 0x63FF2000
03:50:19:CEF-Acct:snaphoting loadinfo 0x63FF1EA0
03:50:19:CEF-Acct:snaphoting loadinfo 0x63FF17C0
03:50:19:CEF-Acct:snaphoting loadinfo 0x63FF1D40
03:50:19:CEF-Acct:snaphoting loadinfo 0x63FF1A80
03:50:19:CEF-Acct:snaphoting loadinfo 0x63FF0740
03:50:19:CEF-Acct:snaphoting loadinfo 0x63FF08A0
03:50:19:CEF-Acct:snaphoting loadinfo 0x63FF0B60
03:50:19:CEF-Acct:snaphoting loadinfo 0x63FF0CC0
03:50:19:CEF-Acct:snaphoting loadinfo 0x63FF0F80
03:50:19:CEF-Acct:snaphoting loadinfo 0x63FF10E0
03:50:19:CEF-Acct:snaphoting loadinfo 0x63FF1240
03:50:19:CEF-Acct:snaphoting loadinfo 0x63FF13A0
03:50:19:CEF-Acct:snaphoting loadinfo 0x63FF1500
03:50:19:CEF-Acct:snaphoting loadinfo 0x63FF1920
03:50:19:CEF-Acct:snaphoting loadinfo 0x63FF0E20
03:50:19:CEF-Acct:snaphoting loadinfo 0x63FF1660
03:50:19:CEF-Acct:snaphoting loadinfo 0x63FF05E0
03:50:19:CEF-Acct:snaphoting loadinfo 0x63FF0A00
03:50:19:CEF-Acct:snaphoting loadinfo 0x63FF1BE0
03:50:19:CEF-Acct:snaphoting loadinfo 0x63FF0480
03:50:19:CEF-Acct:tmstats_binary:aggregation complete, duration 0 seconds
03:50:21:CEF-Acct:tmstats_binary:writing 45 bytes
03:50:24:CEF-Acct:tmstats_binary:writing 45 bytes
03:50:24:CEF-Acct:tmstats_binary:writing 45 bytes
03:50:27:CEF-Acct:tmstats_binary:writing 45 bytes
03:50:29:CEF-Acct:tmstats_binary:writing 45 bytes
.
.
.
03:50:52:CEF-Acct:tmstats_binary:writing 45 bytes
03:50:55:CEF-Acct:tmstats_binary:writing 45 bytes
03:50:57:CEF-Acct:tmstats_binary:writing 45 bytes
03:50:57:CEF-Acct:tmstats_binary:writing 45 bytes
03:50:57:CEF-Acct:tmstats_binary:writing 45 bytes
03:50:57:CEF-Acct:tmstats_binary:writing 45 bytes
03:50:57:CEF-Acct:tmstats_binary:writing 45 bytes
03:50:57:CEF-Acct:tmstats_binary:writing 45 bytes
03:50:57:CEF-Acct:tmstats_binary:writing 45 bytes
03:50:57:CEF-Acct:tmstats_binary:writing 45 bytes
03:50:57:CEF-Acct:tmstats_binary:tmstats file written, status 0
```

Table 50 describes the significant fields shown in the display.

**Table 51          Debug IP CEF Accounting Non-Recursive Field Descriptions**

<b>Field</b>	<b>Description</b>
Beginning generation of tmstats ephemeral file (mode binary)	Tmstats file is being created.
CEF-Acct:snaphoting loadinfo 0x63FF2000	Baseline counters are being written to the tmstats file for each nonrecursive prefix.
CEF-Acct:tmstats_binary:aggregation complete, duration 0 seconds	Tmstats file creation is complete.
CEF-Acct:tmstats_binary:writing 45 bytes	Nonrecursive accounting statistics are being updated to the tmstats file.
CEF-Acct:tmstats_binary:tmstats file written, status 0	Update of the tmstats file is complete.