Debug Command Reference

The products and specifications, configurations, and other technical information regarding the products contained in this manual are subject to change without notice. All statements, technical information, and recommendations contained in this manual are believed to be accurate and reliable but are presented without warranty of any kind, express or implied, and users must take full responsibility for their application of any products specified in this manual.

This equipment generates, uses, and can radiate radio frequency energy and, if not installed and used in accordance with the instruction manual for this device, may cause interference to radio communications. This equipment has been tested and found to comply with the limits for a Class A computing device pursuant to Subpart J of Part 15 of FCC Rules, which are designed to provide reasonable protection against such interference when operated in a commercial environment. Operation of this equipment in a residential area is likely to cause interference, in which case users at their own expense will be required to take whatever measures may be required to correct the interference.

Possession, use, or copying of the software described in this publication is authorized only pursuant to a valid written license from Digital or an authorized sublicensor.

Digital Equipment Corporation makes no representations that the use of its products in the manner described in this publication will not infringe on existing or future patent rights, nor do the descriptions contained in this publication imply the granting of licenses to make, use, or sell equipment or software in accordance with the description.

The following are trademarks of Digital Equipment Corporation: DDCMP, DEC, DECnet, DECNIS, DECserver, DECsystem, DECwindows, Digital, DNA, OpenVMS, ULTRIX, VAX, VAXstation, VMS, VMScluster, and the DIGITAL logo.

Portions of this document is used with permission of Cisco Systems, Incorporated. Copyright © 1990 - 1995, Cisco Systems, Inc.

The following third-party software may be included with your product and will be subject to the software license agreement:

CiscoWorks software and documentation are based in part on HP OpenView under license from the Hewlett-Packard Company. HP OpenView is a trademark of the Hewlett-Packard Company. Copyright © 1992, 1993 Hewlett-Packard Company.

The Cisco implementation of TCP header compression is an adaptation of a program developed by the University of California, Berkeley (UCB) as part of UCB’s public domain version of the UNIX operating system. All rights reserved. Copyright © 1981, Regents of the University of California.

Network Time Protocol (NTP). Copyright © 1992, David L. Mills. The University of Delaware makes no representations about the suitability of this software for any purpose.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

About This Manual

This section introduces the *Debug Command Reference* publication audience and scope, organization, use, and conventions.

Audience and Scope

This publication addresses the network or system administrator who maintains a Cisco gateway, router, or bridge running Internetwork Operating System (IOS) Release 10 and earlier software.

Readers should know how to configure a Cisco router and should be familiar with the protocols and media their routers are configured to support. Readers must also be aware of their network topology.

Document Organization and Use

The *Debug Command Reference* publication provides information about using **debug** commands to troubleshoot Cisco network servers. This manual is most effective when used in conjunction with the *Troubleshooting Internetworking Systems* publication.

Chapter 1, "Using Debug Commands," explains how you enter **debug** commands; use the **debug ?** and **debug all** commands; and generate and redirect **debug** command output. It is important that you read this chapter first before proceeding to Chapter 2, "Debug Commands."

Chapter 2, "Debug Commands," presents reference information on commands you use to debug your internetwork. The chapter includes command function descriptions, sample output displays, and explanations of these displays.

Appendix A, "X.25 Cause and Diagnostic Codes," lists the codes that can appear in output from the **debug x25**, **debug x25-events**, and **debug x25-vc** commands.

Appendix B, "ISDN Switch Types, Codes, and Values," lists the supported switch types. It also contains the cause codes, cause values, bearer capability values, and progress values that can appear in output from the **debug isdn-q921**, **debug isdn-q931**, and **debug isdn-event** commands.

Document Conventions

The command descriptions in this manual use these conventions:

- Commands and keywords are in **boldface**.
- Filenames, directory names, and arguments for which you supply values are in *italics*.
- Elements in square brackets ([ ]) are optional.
- Alternative but required keywords are grouped in braces ({ }) and are separated by vertical bars (|).
- A string is defined as a nonquoted set of characters. For example, when setting up a community string for SNMP to “public,” do not use quotes around the string or the string will be set to “public.”

The samples use these conventions:

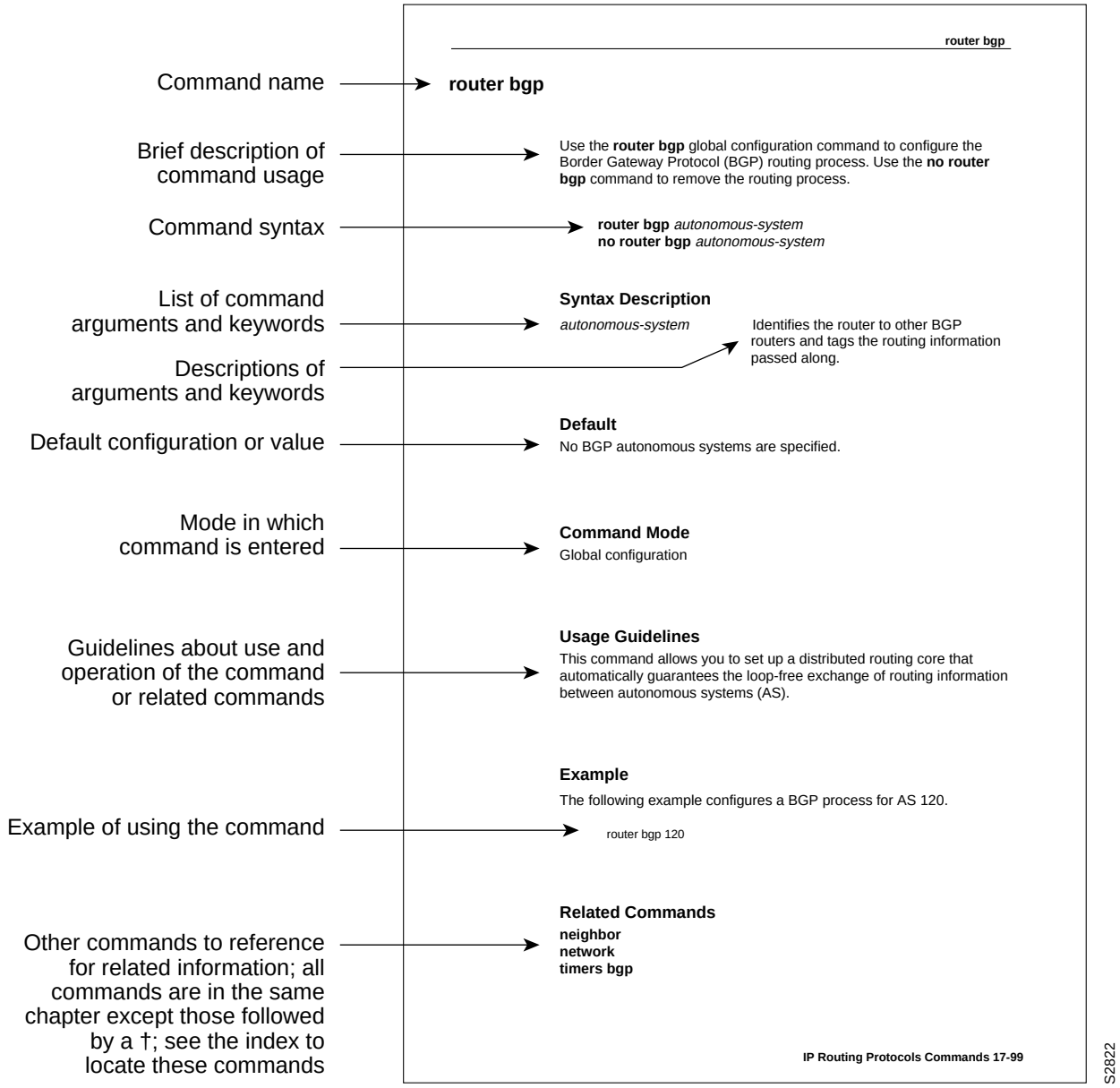
- Terminal sessions are printed in a `screen` font.
- Information you enter is in a **`boldface screen`** font.
- Nonprinting characters are shown in angle brackets (< >).
- Information the system displays is in a `screen` font; default responses are in square brackets ([ ]).

This publication also uses the following conventions:

Note Means *reader take note*. Notes contain helpful suggestions, or reference to materials not covered in this manual.

Caution Means *reader be careful*. In this situation, you might do something that could result in equipment damage or loss of data.

The following illustration explains the fields on a typical command reference page:



CHAPTER 1

Using Debug Commands

This chapter explains how you use **debug** commands to diagnose and resolve internetworking problems. Specifically, it covers the following topics:

- Entering **debug** commands
- Using the **debug ?** command
- Using the **debug all** command
- Generating debugging output
- Redirecting debugging output

Caution Because debugging output is assigned high priority in the CPU process, it can render the system unusable. For this reason, only use **debug** commands to troubleshoot specific problems or during troubleshooting sessions with Cisco technical support staff. Moreover, it is best to use **debug** commands during periods of lower network traffic and fewer users. Debugging during these periods decreases the likelihood that increased **debug** command processing overhead will affect system use.

Entering Debug Commands

All **debug** commands are entered while in privileged EXEC mode and most **debug** commands do not take any arguments. For example, to enable the **debug broadcast** command, enter the following in privileged EXEC mode at the command line:

debug broadcast

To turn off the **debug broadcast** command, in privileged EXEC mode, enter the **no** form of the command at the command line:

no debug broadcast

Alternately, in privileged EXEC mode, you can enter the **undebug** form of the command:

undebug broadcast

To display the state of each debugging option, enter the following at the command line in privileged EXEC mode:

show debugging

Using the Debug ? Command

To list and briefly describe all of the debugging command options, enter the following command in privileged EXEC mode at the command line:

debug ?

Using the Debug All Command

To enable all system diagnostics, enter the following command in privileged EXEC mode at the command line:

debug all

The **no debug all** command turns off all diagnostic output. Using the **no debug all** command is a convenient way to ensure that you have not accidentally left any **debug** commands turned on.

Caution Because debugging output takes priority over other network traffic, and because the **debug all** command generates more output than any other **debug** command, it can severely diminish the router's performance or even render it unusable. In virtually all cases, it is best to use more specific **debug** commands.

Generating Debug Command Output

Enabling a **debug** command can result in output similar to the example shown in Figure 1-1 for the **debug broadcast** command.

Figure 1-1 Example Debug Broadcast Output

```
router# debug broadcast

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

The router continues to generate such output until you enter the corresponding **no debug** command (in this case, **no debug broadcast**).

If you enable a **debug** command and no output is displayed, consider the following possibilities:

- The router may not be properly configured to generate the type of traffic you want to monitor. Use the **write terminal** command to check its configuration.
- Even if the router is properly configured, it may not generate the type of traffic you want to monitor during the particular period that debugging is turned on. Depending on the protocol you are debugging, you can use commands such as the TCP/IP **ping** command to generate network traffic.

Redirecting Debugging and Error Message Output

By default, the network server sends the output from **debug** commands and system error messages to the console terminal. If you use this default, monitor debugging output using a virtual terminal connection, rather than the console port.

To redirect debugging output, use the **logging** command options within configuration mode.

Possible destinations include the console terminal, virtual terminals, internal buffer, and UNIX hosts running a syslog server. The syslog format is compatible with 4.3 BSD UNIX and its derivatives.

Note Be aware that the debugging destination you use affects system overhead. Logging to the console produces very high overhead, whereas logging to a virtual terminal produces less overhead. Logging to a syslog server produces even less, and logging to an internal buffer produces the least overhead of any method.

To configure message logging, you need to be in configuration command mode. To enter this mode, use the **configure terminal** command at the EXEC prompt.

The following sections describe how to select redirection options with the **logging** router configuration command.

Enabling Message Logging

To enable message logging to all supported destinations other than the console, enter the following:

logging on

The default condition is **logging on**.

To direct logging to the console terminal only and disable logging output to other destinations, enter the following command:

no logging on

Setting the Message Logging Levels

You can set the logging levels when logging messages to the following:

- Console
- Monitor
- Syslog server

Table 1-1 lists and briefly describes the logging levels and corresponding keywords you can use to set the logging levels for these types of messages. The highest level of message is level 0, emergencies. The lowest level is level 7, debugging, which also displays the greatest amount of messages. For information about limiting these messages, see sections later in this chapter.

Table 1-1 Message Logging Keywords and Levels

Level	Keyword	Description	Syslog Definition
0	**emergencies**	System is unusable.	LOG_EMERG
1	**alerts**	Immediate action is needed.	LOG_ALERT
2	**critical**	Critical conditions exist.	LOG_CRIT
3	**errors**	Error conditions exist.	LOG_ERR
4	**warnings**	Warning conditions exist.	LOG_WARNING
5	**notification**	Normal, but significant, conditions exist.	LOG_NOTICE
6	**informational**	Informational messages.	LOG_INFO
7	**debugging**	Debugging messages.	LOG_DEBUG

Limiting the Types of Logging Messages Sent to the Console

To limit the types of messages that are logged to the console, use the **logging console** router configuration command. The full syntax of this command follows:

logging console *level*
no logging console

The **logging console** command limits the logging messages displayed on the console terminal to messages up to and including the specified severity level, which is specified by the *level* argument.

The *level* argument can be one of the keywords listed in Table 1-1. They are listed in order from the most severe level to the least severe.

The **no logging console** command disables logging to the console terminal.

Example

The following example sets console logging of messages at the **debugging** level, which is the least severe level and will display all logging messages:

```
logging console debugging
```

Logging Messages to an Internal Buffer

The default logging device is the console; all messages are displayed on the console unless otherwise specified.

To log messages to an internal buffer, use the **logging buffered** router configuration command. The full syntax of this command follows:

logging buffered
no logging buffered

The **logging buffered** command copies logging messages to an internal buffer instead of writing them to the console terminal. The buffer is circular in nature, so newer messages overwrite older messages. To display the messages that are logged in the buffer, use the privileged EXEC command **show logging**. The first message displayed is the oldest message in the buffer.

The **no logging buffered** command cancels the use of the buffer and writes messages to the console terminal (the default).

Limiting the Types of Logging Messages Sent to Another Monitor

To limit the level of messages logged to the terminal lines (monitors), use the **logging monitor** router configuration command. The full syntax of this command follows:

logging monitor *level*
no logging monitor

The **logging monitor** command limits the logging messages displayed on terminal lines other than the console line to messages with a level up to and including the specified *level* argument. The *level* argument is one of the keywords listed in Table 1-1. To display logging messages on a terminal (virtual console), use the privileged EXEC command **terminal monitor**.

The **no logging monitor** command disables logging to terminal lines other than the console line.

Example

The following example sets the level of messages displayed on monitors other than the console to **notification**:

```
logging monitor notification
```

Logging Messages to a UNIX Syslog Server

To log messages to the syslog server host, use the **logging** router configuration command. The full syntax of this command follows:

logging *ip-address*
no logging *ip-address*

The **logging** command identifies a syslog server host to receive logging messages. The *ip-address* argument is the IP address of the host. By issuing this command more than once, you build a list of syslog servers that receive logging messages.

The **no logging** command deletes the syslog server with the specified address from the list of syslogs.

Limiting Messages to a Syslog Server

To limit how many messages are sent to the syslog servers, use the **logging trap** router configuration command. The full syntax of this command follows:

logging trap *level*
no logging trap

The **logging trap** command limits the logging messages sent to syslog servers to messages with a level up to and including the specified *level* argument. The *level* argument is one of the keywords listed in Table 1-1.

To send logging messages to a syslog server, specify its host address with the **logging** command.

The default trap level is **informational**.

The **no logging trap** command disables logging to syslog servers.

The current software generates four categories of syslog messages:

- Error messages about software or hardware malfunctions, displayed at the **errors** level.
- Interface up/down transitions and system restart messages, displayed at the **notification** level.

- Reload requests and low-process stack messages, displayed at the **informational** level.
- Output from the **debug** commands, displayed at the **debugging** level.

The privileged EXEC command **show logging** displays the addresses and levels associated with the current logging setup. The command output also includes ancillary statistics.

Example of Setting Up a UNIX Syslog Daemon

To set up the syslog daemon on a 4.3 BSD UNIX system, include a line such as the following in the file */etc/syslog.conf*:

```
local7.debugging /usr/adm/logs/tiplog
```

The **local7** keyword specifies the logging facility to be used.

The **debugging** keyword specifies the syslog level. See Table 1-1 for other keywords that can be listed.

The UNIX system sends messages at or above this level to the specified file, in this case */usr/adm/logs/tiplog*. The file must already exist, and the syslog daemon must have permission to write to it.

CHAPTER 2

Debug Commands

This chapter contains an alphabetical listing of the **debug** commands. Documentation for each command includes a brief description of its use, command syntax, usage guidelines, sample output, and a description of that output.

Output formats vary with each **debug** command. Some generate a single line of output per packet, whereas others generate multiple lines of output per packet. Some generate large amounts of output; others generate only occasional output. Some generate lines of text, and others generate information in field format. Thus, the way the **debug** commands are documented also varies. For example, for **debug** commands that generate lines of text, the output is described line by line. For **debug** commands that generate output in field format, tables are used to describe the fields.

By default, the network server sends the output from the **debug** commands to the console terminal. Sending output to a terminal (virtual console) produces less overhead than sending it to the console. Use the privileged EXEC command **terminal monitor** to send output to a terminal. For more information about redirecting output, see the "Using Debug Commands" chapter.

debug apple arp

Use the **debug apple arp** EXEC command to enable debugging of the AppleTalk Address Resolution Protocol (AARP). The **no** form of this command disables debugging output.

debug apple arp [*type number*]
no debug apple arp [*type number*]

Syntax Description

type	(Optional) Interface type
number	(Optional) Interface number

Command Mode

EXEC

Usage Guidelines

This command is helpful when you experience problems communicating with a node on the network you control (a neighbor). If the **debug apple arp** display indicates that the router is receiving AARP probes, you can assume that the problem does not reside at the physical layer.

Sample Display

Figure 2-1 shows sample **debug apple arp** output.

Figure 2-1 Sample Debug Apple ARP Output

```
router# debug apple arp

Ether0: AARP: Sent resolve for 4160.26
Ether0: AARP: Reply from 4160.26(0000.0c00.0453) for 4160.154(0000.0c00.8ea9)
Ether0: AARP: Resolved waiting request for 4160.26(0000.0c00.0453)
Ether0: AARP: Reply from 4160.19(0000.0c00.0082) for 4160.154(0000.0c00.8ea9)
Ether0: AARP: Resolved waiting request for 4160.19(0000.0c00.0082)
Ether0: AARP: Reply from 4160.19(0000.0c00.0082) for 4160.154(0000.0c00.8ea9)
```

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

The following line indicates that the router has requested the hardware MAC address of the host at network address 4160.26:

```
Ether0: AARP: Sent resolve for 4160.26
```

The following line indicates that the host at network address 4160.26 has replied, giving its MAC address (0000.0c00.0453). For completeness, the message also shows the network address to which the reply was sent and its hardware MAC address (also in parentheses).

```
Ether0: AARP: Reply from 4160.26(0000.0c00.0453) for 4160.154(0000.0c00.8ea9)
```

The following line indicates that the MAC address request is complete:

```
Ether0: AARP: Resolved waiting request for 4160.26(0000.0c00.0453)
```

debug apple domain

Use the **debug apple domain** EXEC command to enable debugging of the AppleTalk domain lookups. The **no** form of this command disables debugging output.

debug apple domain
no debug apple domain

Syntax Description

This command has no arguments or keywords.

Command Mode

EXEC

Usage Guidelines

Use the **debug apple domain** command to observe activity between domains and subdomains. Use this command in conjunction with the **debug apple remap** command to observe interaction between remapping and domain activity. Messages are displayed when the state of a domain changes, such as creating a new domain, deleting a domain, updating a domain, and creating domain neighbors.

Sample Display

Figure 2-2 shows sample **debug apple domain** output intermixed with output from the **debug apple remap** command; the two commands show related events.

Figure 2-2 Sample Debug Apple Domain Output

```
router# debug apple domain

AT-REMAP: RemapProcess for net 3000 domain Domain 1
AT-REMAP: ReshuffleRemapList for subdomain 1
AT-REMAP:  Could not find a remap for cable 3000-3001
AT-DOMAIN: Disabling Domain 1 [ Domain 1 ]
AT-DOMAIN: Disabling interface Ethernet1
AT-DOMAIN: atdomain_DisablePort for Ethernet1
AT-DOMAIN: CleanUpDomain for domain 1 [Domain 1]
AT-DOMAIN: CleanSubDomain for inbound subdomain 1
AT-REMAP: Remap for net 70 inbound subdomain 1 has been deleted
AT-DOMAIN: DeleteAvRemapList for inbound subdomain 1
AT-DOMAIN: DeleteRemapTable for subdomain 1
AT-DOMAIN: DeleteAvRemapList for inbound subdomain 1
AT-DOMAIN: CleanSubDomain for outbound subdomain 1
AT-DOMAIN: DeleteRemapTable for subdomain 1
AT-REMAP: Remap for net 50 outbound subdomain 1 has been deleted
AT-DOMAIN: DeleteAvRemapList for outbound subdomain 1
AT-DOMAIN: DeleteAvRemapList for outbound subdomain 1
AT-DOMAIN: CleanUpDomain for domain 1 [Domain 1]
AT-DOMAIN: CleanSubDomain for inbound subdomain 1
AT-DOMAIN: DeleteRemapTable for subdomain 1
AT-DOMAIN: DeleteAvRemapList for inbound subdomain 1
AT-DOMAIN: CleanSubDomain for outbound subdomain 1
AT-DOMAIN: DeleteRemapTable for subdomain 1
AT-DOMAIN: DeleteAvRemapList for outbound subdomain 1
```

Most lines of output in Figure 2-2 are from the **debug apple domain** command and are self-explanatory.

Related Commands

debug apple remap

debug apple errors

Use the **debug apple errors** EXEC command to display errors occurring in the AppleTalk network. The **no** form of this command disables debugging output.

debug apple errors [*type number*]
no debug apple errors [*type number*]

Syntax Description

type	(Optional) Interface type
number	(Optional) Interface number

Command Mode

EXEC

Usage Guidelines

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

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

Sample Display

Figure 2-3 shows sample **debug apple errors** output when a router is brought up with a zone that does not agree with the zone list of other routers on the network.

Figure 2-3 Debug Apple Errors Output

```
router# debug apple errors

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

As Figure 2-3 suggests, a single error message indicates zone list incompatibility; this message is sent out periodically until the condition is corrected or **debug apple errors** is turned off.

Most of the other messages that **debug apple errors** can generate are obscure or indicate a serious problem with the AppleTalk network. Some of these other messages follow.

In the following message, RTMPRsp, RTMPReq, ATP, AEP, ZIP, ADSP, or SNMP could replace NBP, and "llap dest not for us" could replace "wrong encapsulation":

```
Packet discarded, src 4160.12-254,dst 4160.19-254,NBP,wrong encapsulation
```

In the following message, in addition to invalid echo packet, other possible errors are unsolicited AEP echo reply, unknown echo function, invalid ping packet, unknown ping function, and bad responder packet type.

```
Ethernet0: AppleTalk packet error; no source address available
AT: pak_reply: dubious reply creation, dst 4160.19
AT: Unable to get a buffer for reply to 4160.19

Processing error, src 4160.12-254,dst 4160.19-254,AEP, invalid echo packet
```

The **debug apple errors** command can print out additional messages when other debugging commands are also turned on. When you turn on both **debug apple errors** and **debug apple events**, the following message can be generated:

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

In the preceding message, in addition to NetInfo Reply format is invalid, other possible errors are NetInfoReply not for me, NetInfoReply ignored, NetInfoReply for operational net ignored, NetInfoReply from invalid port, unexpected NetInfoReply ignored, cannot establish primary zone, no primary has been set up, primary zone invalid, net information mismatch, multicast mismatch, and zones disagree.

When you turn on both **debug apple errors** and **debug apple nbp**, the following message can be generated:

```
Processing error, ...,NBP,NBP name invalid
```

In the preceding message, in addition to NBP name invalid, other possible errors are NBP type invalid, NBP zone invalid, not operational, error handling brrq, error handling proxy, NBP fwdreq unexpected, No route to srcnet, Proxy to "*" zone, Zone "*" from extended net, No zone info for "*", and NBP zone unknown.

When you turn on both **debug apple errors** and **debug apple routing**, the following message can be generated:

```
Processing error, ...,RTMPReq, unknown RTMP request
```

In the preceding message, in addition to unknown RTMP request, other possible errors are RTMP packet header bad, RTMP cable mismatch, routed RTMP data, RTMP bad tuple, and Not Req or Rsp.

debug apple events

Use the **debug apple events** EXEC command to display information about AppleTalk special events, neighbors becoming reachable/unreachable, and interfaces going up/down. Only significant events (for example, neighbor and route changes) are logged. The **no** form of this command disables debugging output.

debug apple events [*type number*]
no debug apple events [*type number*]

Syntax Description

type	(Optional) Interface type
number	(Optional) Interface number

Command Mode

EXEC

Usage Guidelines

The **debug apple events** command is useful for solving AppleTalk network problems because it provides an overall picture of the stability of the network. In a stable network, the **debug apple events** command does not return any information. If the command generates numerous messages, those messages can indicate possible sources of the problems.

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

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

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

Sample Display

Figure 2-4 shows sample **debug apple events** output that describes a nonseed router coming up in discovery mode.

Figure 2-4 Sample Debug Apple Events Output with Discovery Mode State Changes

As Figure 2-4 shows, the **debug apple events** command is useful in tracking the discovery mode state changes through which an interface progresses. When no problems are encountered, the state changes progress as follows:

1. Line down
2. Restarting
3. Probing (for its own address [node ID] using AARP)
4. Acquiring (sending out GetNetInfo requests)
5. Requesting zones (the list of zones for its cable)
6. Verifying (that the router's configuration is correct. If not, a port configuration mismatch is declared.)
7. Checking zones (to make sure its list of zones is correct)
8. Operational (participating in routing)

Explanations for individual lines of output in Figure 2-4 follow.

The following message indicates that a port is set. In this case, the zone multicast address is being reset:

```
Ether0: AT: Resetting interface address filters
```

The following messages indicate that the router is changing to restarting mode:

```
%AT-5-INTRESTART: Ether0: AppleTalk port restarting; protocol restarted
Ether0: AppleTalk state changed; unknown -> restarting
```

The following message indicates that the router is probing in the startup range of network numbers (65280-65534) to discover its network number:

```
Ether0: AppleTalk state changed; restarting -> probing
```

The following message indicates that the router is enabled as a nonrouting node using a provisional network number within its startup range of network numbers. This type of message only appears if the network address the router will use differs from its configured address. This is always the case for a discovery-enabled router; it is rarely the case for a nondiscovery-enabled router.

```
%AT-6-ADDRUSED: Ether0: AppleTalk node up; using address 65401.148
```

The following messages indicate that the router is sending out GetNetInfo requests to discover the default zone name and the actual network number range in which its network number can be chosen:

```
Ether0: AppleTalk state changed; probing -> acquiring
%AT-6-ACQUIREMODE: Ether0: AT port initializing; acquiring net configuration
```

Now that the router has acquired the cable configuration information, the following message indicates that it restarts using that information:

```
Ether0: AppleTalk state changed; acquiring -> restarting
```

The following messages indicate that the router is probing for its actual network address:

```
Ether0: AppleTalk state changed; restarting -> line down
Ether0: AppleTalk state changed; line down -> restarting
Ether0: AppleTalk state changed; restarting -> probing
```

The following message indicates that the router has found an actual network address to use:

```
%AT-6-ADDRUSED: Ether0: AppleTalk node up; using address 4160.148
```

The following messages indicate that the router is sending out GetNetInfo requests to verify the default zone name and the actual network number range from which its network number can be chosen:

```
Ether0: AppleTalk state changed; probing -> acquiring
%AT-6-ACQUIREMODE: Ether0: AT port initializing; acquiring net configuration
```

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

```
Ether0: AppleTalk state changed; acquiring -> requesting zones
```

The following messages indicate that the router is sending out GetNetInfo requests to make sure its understanding of the configuration is correct:

```
Ether0: AppleTalk state changed; requesting zones -> verifying
AT: Sent GetNetInfo request broadcast on Ethernet0
```

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

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

The following message indicates that the router is now fully operational as a routing node and can begin routing:

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

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

Figure 2-5 Sample Debug Apple Events Output Showing Seed Coming Up by Itself

```
router# debug apple events

Ethernet1: AT: Resetting interface address filters
%AT-5-INTRESTART: Ethernet1: AppleTalk port restarting; protocol restarted
Ethernet1: AppleTalk state changed; unknown -> restarting
Ethernet1: AppleTalk state changed; restarting -> probing
%AT-6-ADDRUSED: Ethernet1: AppleTalk node up; using address 4165.204
Ethernet1: AppleTalk state changed; probing -> verifying
AT: Sent GetNetInfo request broadcast on Ethernet1
Ethernet1: AppleTalk state changed; verifying -> operational
%AT-6-ONLYROUTER: Ethernet1: AppleTalk port enabled; no neighbors found
```

S2543

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

As Figure 2-5 shows, a nondiscovery-enabled router can come up when no other router is on the wire; however, it must assume that its configuration (if accurate syntactically) is correct, because no other router can verify it. Notice that the last line in Figure 2-5 indicates this situation.

Figure 2-6 shows sample **debug apple events** output that describes a discovery-enabled router coming up when there is no seed router on the wire.

Figure 2-6 Debug Apple Events Output Showing Nonseed with No Seed

```
router# debug apple events

Ether0: AT: Resetting interface address filters
%AT-5-INTRESTART: Ether0: AppleTalk port restarting; protocol restarted
Ether0: AppleTalk state changed; unknown -> restarting
Ether0: AppleTalk state changed; restarting -> probing
%AT-6-ADDRUSED: Ether0: AppleTalk node up; using address 65401.148
Ether0: AppleTalk state changed; probing -> acquiring
AT: Sent GetNetInfo request broadcast on Ether0
AT: Sent GetNetInfo request broadcast on Ether0
AT: Sent GetNetInfo request broadcast on Ether0
AT: Sent GetNetInfo request broadcast on Ether0
AT: Sent GetNetInfo request broadcast on Ether0
```

As Figure 2-6 shows, when you attempt to bring up a nonseed router without a seed router on the wire, it never becomes operational; instead, it hangs in the acquiring mode and continues to send out periodic GetNetInfo requests.

Figure 2-7 shows sample **debug apple events** output when a nondiscovery-enabled router is brought up on an AppleTalk internetwork that is in compatibility mode (set up to accommodate extended as well as nonextended AppleTalk) and the router has violated internetwork compatibility.

Figure 2-7 Sample Debug Apple Events Output Showing Compatibility Conflict

```
router# debug apple events

E0: AT: Resetting interface address filters
%AT-5-INTRESTART: E0: AppleTalk port restarting; protocol restarted
E0: AppleTalk state changed; restarting -> probing
%AT-6-ADDRUSED: E0: AppleTalk node up; using address 41.19
E0: AppleTalk state changed; probing -> verifying
AT: Sent GetNetInfo request broadcast on Ethernet0
%AT-3-ZONEDISAGREES: E0: AT port disabled; zone list incompatible with 41.19
AT: Config error for E0, primary zone invalid
E0: AppleTalk state changed; verifying -> config mismatch
```

S2545

Indicates configuration mismatch

The three configuration command lines that follow indicate the part of the router's configuration that caused the configuration mismatch shown in Figure 2-7:

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

The router shown in Figure 2-7 had been configured with a cable range of 41-41 instead of 40-40, which would have been accurate. Additionally, the zone name was configured incorrectly; it should have been "Marketing," rather than being misspelled as "Marketign."

debug apple nbp

Use the **debug apple nbp** EXEC command to display debugging output from the Name Binding Protocol (NBP) routines. The **no** form of this command disables debugging output.

debug apple nbp [*type number*]
no debug apple nbp [*type number*]

Syntax Description

type	(Optional) Interface type
number	(Optional) Interface number

Command Mode

EXEC

Usage Guidelines

To determine whether the router is receiving NBP lookups from a node on the AppleTalk network, enable **debug apple nbp** at each node between the router and the node in question to determine where the problem lies.

Note Because the **debug apple nbp** command can generate many messages, use it only when the router's CPU utilization is less than 50 percent.

Sample Display

Figure 2-8 shows sample **debug apple nbp** output.

Figure 2-8 Sample Debug Apple NBP Output

```
router# debug apple nbp

AT: NBP ctrl = LkUp, ntuples = 1, id = 77
AT: 4160.19, skt 2, enum 0, name: =:ciscoRouter@Low End SW Lab
AT: LkUp =:ciscoRouter@Low End SW Lab

AT: NBP ctrl = LkUp-Reply, ntuples = 1, id = 77
AT: 4160.154, skt 254, enum 1, name: lestat.Ether0:ciscoRouter@Low End SW Lab

AT: NBP ctrl = LkUp, ntuples = 1, id = 78
AT: 4160.19, skt 2, enum 0, name: =:IPADDRESS@Low End SW Lab
AT: NBP ctrl = LkUp, ntuples = 1, id = 79
AT: 4160.19, skt 2, enum 0, name: =:IPGATEWAY@Low End SW Lab
AT: NBP ctrl = LkUp, ntuples = 1, id = 83
AT: 4160.19, skt 2, enum 0, name: =:ciscoRouter@Low End SW Lab
AT: LkUp =:ciscoRouter@Low End SW Lab

AT: NBP ctrl = LkUp, ntuples = 1, id = 84
AT: 4160.19, skt 2, enum 0, name: =:IPADDRESS@Low End SW Lab

AT: NBP ctrl = LkUp, ntuples = 1, id = 85
AT: 4160.19, skt 2, enum 0, name: =:IPGATEWAY@Low End SW Lab
AT: NBP ctrl = LkUp, ntuples = 1, id = 85
AT: 4160.19, skt 2, enum 0, name: =:IPGATEWAY@Low End SW Lab
```

The first three lines in Figure 2-8 describe an NBP lookup request:

```
AT: NBP ctrl = LkUp, ntuples = 1, id = 77
AT: 4160.19, skt 2, enum 0, name: =:ciscoRouter@Low End SW Lab
AT: LkUp =:ciscoRouter@Low End SW Lab
```

Table 2-1 describes the fields in the first line of output shown in Figure 2-8.

Table 2-1 Debug Apple NBP Field Descriptions—Part 1

Field	Description
AT: NBP	Indicates that this message describes an AppleTalk NBP packet.
ctrl = LkUp	Identifies the type of NBP packet. Possible values include LkUp—NBP lookup request. LkUp-Reply—NBP lookup reply.
ntuples = 1	Indicates the number of name-address pairs in the lookup request packet. Range: 1-31 tuples.
id = 77	Identifies an NBP lookup request value.

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

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

Field	Description
AT:	Indicates that this message describes an AppleTalk packet.
4160.19	Indicates the network address of the requester.
skt 2	Indicates the internet socket address of the requester. The responder will send the NBP lookup reply to this socket address.
enum 0	Indicates the enumerator field. Used to identify multiple names registered on a single socket. Each tuple is assigned its own enumerator, incrementing from 0 for the first tuple.
name: =:ciscoRouter@Low End SW Lab	Indicates the entity name for which a network address has been requested. The AppleTalk entity name includes three components: Object (in this case, a wildcard character (=), indicating that the requester is requesting name-address pairs for all objects of the specified type in the specified zone) Type (in this case, ciscoRouter) Zone (in this case, Low End SW Lab)

The third line in Figure 2-8 essentially reiterates the information in the two lines above it, indicating that a lookup request has been made regarding name-address pairs for all objects of the ciscoRouter type in the Low End SW Lab zone.

Because the router is defined as an object of type ciscoRouter in zone Low End SW Lab, the router sends an NBP lookup reply in response to this NBP lookup request. The following two lines of output from Figure 2-8 show the router's response:

```
AT: NBP ctrl = LkUp-Reply, ntuples = 1, id = 77
AT: 4160.154, skt 254, enum 1, name: lestat.Ether0:ciscoRouter@Low End SW Lab
```

In the first line, ctrl = LkUp-Reply identifies this NBP packet as an NBP lookup request. The same value in the id field (id = 77) associates this lookup reply with the previous lookup request. The second line indicates that the network address associated with the router's entity name (lestat.Ether0:ciscoRouter@Low End SW Lab) is 4160.154. The fact that no other entity name/network address is listed indicates that the responder only knows about itself as an object of type ciscoRouter in zone Low End SW Lab.

debug apple packet

Use the **debug apple packet** EXEC command to display per-packet debugging output. The output reports information online when a packet is received or a transmit is attempted. The **no** form of this command disables debugging output.

debug apple packet [*type number*]
no debug apple packet [*type number*]

Syntax Description

type	(Optional) Interface type
number	(Optional) Interface number

Command Mode

EXEC

Usage Guidelines

With this command, you can monitor the types of packets being slow switched. It displays at least one line of debugging output per AppleTalk packet processed.

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

When invoked in conjunction with the **debug apple errors** command, the **debug apple packet** command reports packet-level problems, such as those concerning encapsulation.

Note Because the **debug apple packet** command can generate many messages, use it only when the router's CPU utilization is less than 50 percent.

Sample Display

Figure 2-9 shows sample **debug apple packet** output.

Figure 2-9 Sample Debug Apple Packet Output

```
router# debug apple packet

Ether0: AppleTalk packet: enctype SNAP, size 60, encaps000000000000000000000000
AT: src=Ethernet0:4160.47, dst=4160-4160, size=10, 2 rtes, RTMP pkt sent
AT: ZIP Extended reply rcvd from 4160.19
AT: ZIP Extended reply rcvd from 4160.19
AT: src=Ethernet0:4160.47, dst=4160-4160, size=10, 2 rtes, RTMP pkt sent
Ether0: AppleTalk packet: enctype SNAP, size 60, encaps000000000000000000000000
Ether0: AppleTalk packet: enctype SNAP, size 60, encaps000000000000000000000000
```

Table 2-3 describes the fields in the first line of output shown in Figure 2-9.

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

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

Table 2-4 describes the fields in the second line of output shown in Figure 2-9.

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

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

The third line in Figure 2-9 indicates the type of packet received and its source AppleTalk address. This message is repeated in the fourth line because AppleTalk hosts can send multiple replies to a given GetNetInfo request.

debug apple remap

Use the **debug apple remap** EXEC command to enable debugging of the AppleTalk remap lookups. The **no** form of this command disables debugging output.

debug apple remap
no debug apple remap

Syntax Description

This command has no arguments or keywords.

Command Mode

EXEC

Usage Guidelines

Use the **debug apple remap** command with the **debug apple domain** command to observe activity between domains and subdomains. Messages from **debug apple remap** are displayed when a particular remapping function occurs, such as creating remaps or deleting remaps.

Sample Display

Figure 2-10 shows sample **debug apple remap** output intermixed with output from the **debug apple domain** command; the two commands show related events.

Figure 2-10 Sample Debug Output

```
router# debug apple remap
router# debug apple domain

AT-REMAP: RemapProcess for net 3000 domain Domain 1
AT-REMAP: ReshuffleRemapList for subdomain 1
AT-REMAP:  Could not find a remap for cable 3000-3001
AT-DOMAIN: Disabling Domain 1 [ Domain 1 ]
AT-DOMAIN: Disabling interface Ethernet1
AT-DOMAIN: atdomain_DisablePort for Ethernet1
AT-DOMAIN: CleanUpDomain for domain 1 [Domain 1]
AT-DOMAIN: CleanSubDomain for inbound subdomain 1
AT-REMAP: Remap for net 70 inbound subdomain 1 has been deleted
AT-DOMAIN: DeleteAvRemapList for inbound subdomain 1
AT-DOMAIN: DeleteRemapTable for subdomain 1
AT-DOMAIN: DeleteAvRemapList for inbound subdomain 1
AT-DOMAIN: CleanSubDomain for outbound subdomain 1
AT-DOMAIN: DeleteRemapTable for subdomain 1
AT-REMAP: Remap for net 50 outbound subdomain 1 has been deleted
AT-DOMAIN: DeleteAvRemapList for outbound subdomain 1
AT-DOMAIN: DeleteAvRemapList for outbound subdomain 1
AT-DOMAIN: CleanUpDomain for domain 1 [Domain 1]
AT-DOMAIN: CleanSubDomain for inbound subdomain 1
AT-DOMAIN: DeleteRemapTable for subdomain 1
AT-DOMAIN: DeleteAvRemapList for inbound subdomain 1
AT-DOMAIN: CleanSubDomain for outbound subdomain 1
AT-DOMAIN: DeleteRemapTable for subdomain 1
AT-DOMAIN: DeleteAvRemapList for outbound subdomain 1
```

Most lines of output in Figure 2-10 are from the **debug apple domain** command. The output from the **debug apple remap** command is self-explanatory.

Related Command

debug apple domain

debug apple routing

Use the **debug apple routing** EXEC command to enable debugging output from the Routing Table Maintenance Protocol (RTMP) routines. The **no** form of this command disables debugging output.

debug apple routing [*type number*]
no debug apple routing [*type number*]

Syntax Description

type	(Optional) Interface type
number	(Optional) Interface number

Command Mode

EXEC

Usage Guidelines

This command can be used to monitor acquisition of routes, aging of routing table entries, and advertisement of known routes. It also reports conflicting network numbers on the same network if the network is misconfigured.

Note Because the **debug apple routing** command can generate many messages, use it only when the router's CPU utilization is less than 50 percent.

Sample Display

Figure 2-11 shows sample **debug apple routing** output.

Figure 2-11 Sample Debug Apple Routing Output

```
router# debug apple routing

AT: src=Ethernet0:4160.41, dst=4160-4160, size=19, 2 rtes, RTMP pkt sent
AT: src=Ethernet1:41069.25, dst=41069, size=427, 96 rtes, RTMP pkt sent
AT: src=Ethernet2:4161.23, dst=4161-4161, size=427, 96 rtes, RTMP pkt sent
AT: Route ager starting (97 routes)
AT: Route ager finished (97 routes)
AT: RTMP from 4160.19 (new 0,old 94,bad 0,ign 0, dwn 0)
AT: RTMP from 4160.250 (new 0,old 0,bad 0,ign 2, dwn 0)
AT: RTMP from 4161.236 (new 0,old 94,bad 0,ign 1, dwn 0)
AT: src=Ethernet0:4160.41, dst=4160-4160, size=19, 2 rtes, RTMP pkt sent
```

Explanations for representative lines of the **debug apple routing** output in Figure 2-11 follow.

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

Table 2-5 Debug Apple Routing Field Descriptions—Part 1

Field	Description
AT:	Indicates that this is AppleTalk debugging output
src = Ethernet0:4160.41	Indicates the source router interface and network address for the RTMP update packet
dst = 4160-4160	Indicates the destination network address for the RTMP update packet
size = 19	Shows the size of this RTMP packet (in bytes)
2 rtes	Indicates that this RTMP update packet includes information on two routes
RTMP pkt sent	Indicates that this type of message describes an RTMP update packet that the router has sent (rather than one that it has received)

The following two messages indicate that the ager has started and finished the aging process for the routing table and that this table contains 97 entries.

```
AT: Route ager starting (97 routes)
AT: Route ager finished (97 routes)
```

Table 2-6 describes the fields in the following line of **debug apple routing** output.

```
AT: RTMP from 4160.19 (new 0,old 94,bad 0,ign 0, dwn 0)
```

Table 2-6 Debug Apple Routing Field Descriptions—Part 2

Field	Description
AT:	Indicates that this is AppleTalk debugging output
RTMP from 4160.19	Indicates the source address of the RTMP update the router received
new 0	Shows the number of routes in this RTMP update packet that the router did not already know about
old 94	Shows the number of routes in this RTMP update packet that the router already knew about
bad 0	Shows the number of routes the other router indicates have gone bad
ign 0	Shows the number of routes the other router ignores
dwn 0	Shows the number of poisoned tuples included in this packet

debug apple zip

Use the **debug apple zip** EXEC command to display debugging output from the Zone Information Protocol (ZIP) routines. The **no** form of this command disables debugging output.

debug apple zip [*type number*]
no debug apple zip [*type number*]

Syntax Description

type	(Optional) Interface type
number	(Optional) Interface number

Command Mode

EXEC

Usage Guidelines

This command reports significant events such as the discovery of new zones and zone list queries. It generates information similar to that generated by **debug apple routing**, but generates it for ZIP packets instead of RTMP packets.

You can use he **debug apple zip** command to determine whether a ZIP storm is taking place in the AppleTalk network. You can detect the existence of a ZIP storm when you see that no router on a cable has the zone name corresponding to a network number that all the routers have in their routing tables.

Sample Display

Figure 2-12 shows sample **debug apple zip** output.

Figure 2-12 Sample Debug Apple ZIP Output

```
router# debug apple zip

AT: Sent GetNetInfo request broadcast on Ether0
AT: Recvd ZIP cmd 6 from 4160.19-6
AT: 3 query packets sent to neighbor 4160.19
AT: 1 zones for 31902, ZIP XReply, src 4160.19
AT: net 31902, zonelen 10, name US-Florida
```

Explanations of the lines of output shown in Figure 2-12 follow.

The first line indicates that the router has received an RTMP update that includes a new network number and is now requesting zone information:

```
AT: Sent GetNetInfo request broadcast on Ether0
```

The second line indicates that the neighbor at address 4160.19 replies to the zone request with a default zone:

```
AT: Recvd ZIP cmd 6 from 4160.19-6
```

The third line indicates that the router responds with three queries to the neighbor at network address 4160.19 for other zones on the network:

```
AT: 3 query packets sent to neighbor 4160.19
```

The fourth line indicates that the neighbor at network address 4160.19 responds with a ZIP extended reply, indicating that one zone has been assigned to network 31902:

```
AT: 1 zones for 31902, ZIP XReply, src 4160.19
```

The fifth line indicates that the router responds that the zone name of network 31902 is US-Florida, and the zone length of that zone name is 10:

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

debug arp

Use the **debug arp** EXEC command to display information on Address Resolution Protocol (ARP) transactions. The **no** form of this command disables debugging output.

debug arp
no debug arp

Syntax Description

This command has no arguments or keywords.

Command Mode

EXEC

Usage Guidelines

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

Sample Display

Figure 2-13 shows sample **debug arp** output.

Figure 2-13 Sample Debug ARP Output

```
router# debug arp

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

In Figure 2-13, each line of output represents an ARP packet that the router sent or received. Explanations for the individual lines of output follow.

The first line indicates that the router at IP address 131.108.22.7 and MAC address 0000.0c01.e117 sent an ARP request for the MAC address of the host at 131.108.22.96. The series of zeros (0000.0000.0000) following this address indicate that the router is currently unaware of the MAC address.

```
IP ARP: sent req src 131.108.22.7 0000.0c01.e117, dst 131.108.22.96 \
0000.0000.0000
```

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

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

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

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

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

```
IP ARP: rep filtered src 131.108.22.7 aa92.1b36.a456, dst 255.255.255.255 \
ffff.ffff.ffff
```

The fifth line indicates that another host on the network attempted to inform the router that it is on network 131.108.9.7, but the router does not know that that network is attached to a different router interface. The remote host (probably a PC or an X terminal) is misconfigured. If the router were to install this entry, it would deny service to the real machine on the proper cable.

```
IP ARP: rep filtered src 131.108.9.7 0000.0c00.6b31, dst 131.108.22.7 \
0800.2010.b908
```

debug atm errors

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

debug atm errors
no debug atm errors

Syntax Description

This command has no arguments or keywords.

Command Mode

EXEC

Sample Display

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

Figure 2-14 Sample Debug ATM Errors Output

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

The line of output in Figure 2-14 indicates that a packet was routed to the ATM interface, but no static map was set up to route that packet to the proper virtual circuit.

debug atm events

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

debug atm events
no debug atm events

Syntax Description

This command has no arguments or keywords.

Command Mode

EXEC

Usage Guidelines

This command displays ATM events that occur on the ATM interface processor and is useful for diagnosing problems in an ATM network. It provides an overall picture of the stability of the network. In a stable network, the **debug atm events** command does not return any information. If the command generates numerous messages, the messages can indicate the possible source of problems.

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

Sample Display

Figure 2-15 shows sample **debug atm events** output.

Figure 2-15 Sample Debug ATM Events Output

```
router# debug atm events
ATM events debugging is on
RESET(ATM4/0): PLIM type is 1, Rate is 100Mbps
aip_disable(ATM4/0): state=1
config(ATM4/0)
aip_love_note(ATM4/0): asr=0x201
aip_enable(ATM4/0)
aip_love_note(ATM4/0): asr=0x4000
aip_enable(ATM4/0): restarting VCs: 7
aip_setup_vc(ATM4/0): vc:1 vpi:1 vci:1
aip_love_note(ATM4/0): asr=0x200
aip_setup_vc(ATM4/0): vc:2 vpi:2 vci:2
aip_love_note(ATM4/0): asr=0x200
aip_setup_vc(ATM4/0): vc:3 vpi:3 vci:3
aip_love_note(ATM4/0): asr=0x200
aip_setup_vc(ATM4/0): vc:4 vpi:4 vci:4
aip_love_note(ATM4/0): asr=0x200
aip_setup_vc(ATM4/0): vc:6 vpi:6 vci:6
aip_love_note(ATM4/0): asr=0x200
aip_setup_vc(ATM4/0): vc:7 vpi:7 vci:7
aip_love_note(ATM4/0): asr=0x200
aip_setup_vc(ATM4/0): vc:11 vpi:11 vci:11
aip_love_note(ATM4/0): asr=0x200
```

Table 2-7 describes significant fields in the output shown in Figure 2-15.

Table 2-7 Debug ATM Events Field Descriptions

Field	Description
PLIM type	Indicates the interface rate in Mbps. Possible values are 1 = TAXI(4B5B) 100 Mbps 2 = SONET 155 Mbps 3 = E3 34 Mbps
state	Indicates current state of the AIP. Possible values are 1 = An ENABLE will be issued soon 0 = The AIP will remain shut down
asr	Defines a bitmask, which indicates actions or completions to commands. Valid bitmask values are 0x0800 = AIP crashed, reload may be required. 0x0400 = AIP detected a carrier state change. 0x0n00 = Command completion status. Command completion status codes are n = 8 Invalid PLIM detected n = 4 Command failed n = 2 Command completed successfully n = 1 CONFIG request failed n = 0 Invalid value

Explanations for representative lines of output in Figure 2-15 follow.

The following line indicates that the ATM Interface Processor (AIP) was reset. The PLIM TYPE detected was 1, so the maximum rate is set to 100 Mbps.

```
RESET(ATM4/0): PLIM type is 1, Rate is 100Mbps
```

The following line indicates that the ATM Interface Processor (AIP) was given a **shutdown** command, but the current configuration indicates that the AIP should be up:

```
aip_disable(ATM4/0): state=1
```

The following line indicates that a configuration command has been completed by the AIP:

```
aip_love_note(ATM4/0): asr=0x201
```

The following line indicates that the AIP was given a **no shutdown** command to take it out of shutdown:

```
aip_enable(ATM4/0)
```

The following line indicates that the AIP detected a carrier state change. It does not indicate that the carrier is down or up, only that it has changed:

```
aip_love_note(ATM4/0): asr=0x4000
```

The following line of output indicates that the AIP enable function is restarting all PVCs automatically:

```
aip_enable(ATM4/0): restarting VCs: 7
```

The following lines of output indicate that PVC 1 was set up and a successful completion code was returned:

```
aip_setup_vc(ATM4/0): vc:1 vpi:1 vci:1
aip_love_note(ATM4/0): asr=0x200
```

debug atm packet

Use the **debug atm packet** EXEC command to display per-packet debugging output. The output reports information online when a packet is received or a transmit is attempted. The **no** form of this command disables debugging output.

debug atm packet
no debug atm packet

Syntax Description

This command has no arguments or keywords.

Command Mode

EXEC

Usage Guidelines

The **debug atm packet** command displays all process-level ATM packets for both outbound and inbound packets. This command is useful for determining whether packets are being received and transmitted correctly.

For transmitted packets, the information is displayed only after the protocol data unit (PDU) is entirely encapsulated and a next hop virtual circuit (VC) is found. If information is not displayed, the address translation probably failed during encapsulation. When a next hop VC is found, the packet is displayed exactly as it will be presented on the wire. Having a display indicates the packets are properly encapsulated for transmission.

For received packets, information is displayed for all incoming frames. The display can show whether the transmitting station properly encapsulates the frames. Because all incoming frames are displayed, this information is useful when performing back-to-back testing and corrupted frames cannot be dropped by an intermediary ATM switch.

The **debug atm packet** command also displays the initial bytes of the actual PDU in hexadecimal. This information can be decoded only by qualified support or engineering personnel.

Note Because the **debug atm packet** command generates a significant amount of output for every packet processed, use it only when traffic on the network is low, so other activity on the system is not adversely affected.

Sample Display

Figure 2-16 shows sample **debug atm packet** output.

Figure 2-16 Sample Debug ATM Packet Output

```
router# debug atm packets
ATM packets debugging is on
router#
ATM2/0(O): VCD: 0x1,DM: 1C00, MUX, ETYPE: 0800,Length: 32
4500 002E 0000 0000 0209 92ED 836C A26E FFFF FFFF 1108 006D 0001 0000 0000
A5CC 6CA2 0000 000A 0000 6411 76FF 0100 6C08 00FF FFFF 0003 E805 DCFF 0105
```

Table 2-8 describes significant fields shown in Figure 2-16.

Table 2-8 Debug ATM Packet Field Descriptions

Field	Description
ATM2/0	Indicates the interface that generated this packet.
(O)	Indicates an output packet. (I) would mean receive packet.
VCD: 0x*n*	Indicates the virtual circuit associated with this packet, where *n* is some value.
DM: 0x*nnnn*	Indicates the descriptor mode bits on output only, where *nnnn* is a hexadecimal value.
ETYPE: *n*	Shows the Ethernet type for this packet.
Length: *n*	Shows the total length of the packet including the ATM header(s).

The following two lines of output are the binary data, which are the contents of the protocol PDU before encapsulation at the ATM:

```
4500 002E 0000 0000 0209 92ED 836C A26E FFFF FFFF 1108 006D 0001 0000 0000
A5CC 6CA2 0000 000A 0000 6411 76FF 0100 6C08 00FF FFFF 0003 E805 DCFF 0105
```

debug bri

Use the **debug bri** EXEC command to display debugging information on Integrated Services Digital Networks (ISDN) Basic Rate Interface (BRI) routing activity. The **no** form of this command disables debugging output.

debug bri
no debug bri

Syntax Description

This command has no arguments or keywords.

Command Mode

EXEC

Usage Guidelines

The **debug bri** command indicates whether the ISDN code is enabling and disabling the B-channels when attempting an outgoing call. This command is available for the low-end router products that have a multi-BRI network interface module installed.

Note Because the **debug bri** command generates a significant amount of output, use it only when traffic on the IP network is low, so other activity on the system is not adversely affected.

Sample Display

Figure 2-17 shows sample **debug bri** output.

Figure 2-17 Sample Debug BRI Packets Output

```
Router# debug bri

Basic Rate network interface debugging is on
BRI: write_sid: wrote 1B for subunit 0, slot 1.
BRI: write_sid: wrote 15 for subunit 0, slot 1.
BRI: write_sid: wrote 17 for subunit 0, slot 1.
BRI: write_sid: wrote 6 for subunit 0, slot 1.
BRI: write_sid: wrote 8 for subunit 0, slot 1.
BRI: write_sid: wrote 11 for subunit 0, slot 1.
BRI: write_sid: wrote 13 for subunit 0, slot 1.
BRI: write_sid: wrote 29 for subunit 0, slot 1.
BRI: write_sid: wrote 1B for subunit 0, slot 1.
BRI: write_sid: wrote 15 for subunit 0, slot 1.
BRI: write_sid: wrote 17 for subunit 0, slot 1.
BRI: write_sid: wrote 20 for subunit 0, slot 1.
BRI: Starting Power Up timer for unit = 0.
BRI: write_sid: wrote 3 for subunit 0, slot 1.
BRI: Starting T3 timer after expiry of PUP timeout for unit = 0, current state is F4.
BRI: write_sid: wrote FF for subunit 0, slot 1.
BRI: Activation for unit = 0, current state is F7.
BRI: enable channel B1
BRI: write_sid: wrote 14 for subunit 0, slot 1.
```

```
%LINK-3-UPDOWN: Interface BRI0: B-Channel 1, changed state to up
%LINK-5-CHANGED: Interface BRI0: B-Channel 1, changed state to up.!!!
BRI: disable channel B1
BRI: write_sid: wrote 15 for subunit 0, slot 1.

%LINK-3-UPDOWN: Interface BRI0: B-Channel 1, changed state to down
%LINK-5-CHANGED: Interface BRI0: B-Channel 1, changed state to down
%LINEPROTO-5-UPDOWN: Line protocol on Interface BRI0: B-Channel 1, changed state to down
```

Explanations for individual lines of output from Figure 2-17 follow.

The following line indicates that an internal command was written to the interface controller. The subunit identifies the first interface in the slot:

```
BRI: write_sid: wrote 1B for subunit 0, slot 1.
```

The following line indicates that the power-up timer was started for the named unit:

```
BRI: Starting Power Up timer for unit = 0.
```

The following lines indicate that the channel or the protocol on the interface changed state:

```
%LINK-3-UPDOWN: Interface BRI0: B-Channel 1, changed state to up
%LINK-5-CHANGED: Interface BRI0: B-Channel 1, changed state to up.!!!
%LINEPROTO-5-UPDOWN: Line protocol on Interface BRI0: B-Channel 1, changed state to down
```

The following line indicates that the channel was disabled:

```
BRI: disable channel B1
```

Lines of output not described are for use by support staff only.

Related Commands

debug isdn-event
debug isdn-q921
debug isdn-q931

debug broadcast

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

debug broadcast
no debug broadcast

Syntax Description

This command has no arguments or keywords.

Command Mode

EXEC

Usage Guidelines

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

Sample Display

Figure 2-18 shows sample **debug broadcast** output. Notice how similar it is to the **debug packet** output.

Figure 2-18 Sample Debug Broadcast Output

```
router# debug broadcast

Ethernet0: Broadcast ARPA, src 0000.0c00.6fa4, dst ffff.ffff.ffff, type 0x0800,
data 45000028000000000FF11EA7B, len 60
Serial3: Broadcast HDLC, size 64, type 0x800, flags 0x8F00
Serial2: Broadcast PPP, size 128
Serial7: Broadcast FRAME-RELAY, size 174, type 0x800, DLCI 7a
```

Table 2-9 describes significant fields shown in Figure 2-18.

Table 2-9 Debug Broadcast Field Descriptions

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

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

debug cdp

Use the **debug cdp** EXEC command to enable debugging of Cisco Discovery Protocol (CDP). The **no** form of this command disables debugging output.

debug cdp {**packets** | **adjacency** | **events**}
no debug cdp {**packets** | **adjacency** | **events**}

Syntax Description

packets	Enables packet-related debugging output.
adjacency	Enables adjacency-related debugging output.
events	Enables output related related to error messages, such as detecting a bad checksum.

Command Mode

EXEC

Usage Guidelines

Use **debug cdp** commands to display information about CDP packet activity, activity between CDP neighbors, and various CDP events.

Sample Display

Figure 2-19 shows a composite sample output from **debug cdp packets**, **debug cdp adjacency**, and **debug cdp events**.

Figure 2-19 Sample Debug CDP Output

```
router# debug cdp packets
CDP packet info debugging is on
router# debug cdp adjacency
CDP neighbor info debugging is on
router# debug cdp events
CDP events debugging is on

CDP-PA: Packet sent out on Ethernet0
CDP-PA: Packet received from gray.cisco.com on interface Ethernet0

CDP-AD: Deleted table entry for violet.cisco.com, interface Ethernet0
CDP-AD: Interface Ethernet2 coming up

CDP-EV: Encapsulation on interface Serial2 failed
```

The messages displayed by **debug cdp** commands are self-explanatory.

debug channel events

The **debug channel events** EXEC command displays processing events that occur on the channel adapter interfaces of all installed adapters. This command is valid for the Cisco 7000 series routers only. The **no** form of this command disables debugging output.

debug channel events
no debug channel events

Syntax Description

This command has no arguments or keywords.

Command Mode

EXEC

Usage Guidelines

This command displays Channel Interface Processor (CIP) events that occur on the CIP interface processor and is useful for diagnosing problems in an IBM channel attach network. It provides an overall picture of the stability of the network. In a stable network, the **debug channel events** command does not return any information except for a statistic message (cip_love_letter) transmitted every ten seconds. If the command generates numerous messages, they can indicate the possible source of the problems.

When configuring or making changes to a router or interface that supports IBM channel attach, enable **debug channel events**. Doing so alerts you to the progress of the changes or to any errors that might result. Also use this command periodically when you suspect network problems.

Sample Display

Figure 2-20 shows sample **debug channel events** output.

Figure 2-20 Sample Debug Channel Events Output

```
Router# debug channel events
Channel3/1: love letter received, bytes 3308
Channel3/0: love letter received, bytes 3336
cip_love_letter: recieved ll, but no cip_info
Channel3/0: cip_reset(), state administratively down
Channel3/0: cip_reset(), state up
Channel3/0: sending nodeid
Channel3/0: sending command for vc 0, CLAW path C700, device C0
```

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

The following line indicates that data was received on the CIP:

```
Channel3/1: love letter received, bytes 3308
```

The following line indicates that the interface is enabled, but there is no configuration for it. It does not normally indicate a problem, just that the route processor (RP) got statistics from the CIP but has no place to store them.

```
cip_love_letter: recieved ll, but no cip_info
```

The following line indicates that the CIP is being reset to an administrative down state:

```
Channel3/0: cip_reset(), state administratively down
```

The following line indicates that the CIP is being reset to an administrative up state:

```
Channel3/0: cip_reset(), state up
```

The following line indicates that the node id is being sent to the CIP. This information is the same as the "Local Node" information under the **show extended channel slot/port subchannels** command. The CIP needs this information to send to the host mainframe.

```
Channel3/0: sending nodeid
```

The following line indicates that a CLAW subchannel command is being sent from the RP to the CIP. The value vc 0 indicates that the CIP will use virual circuit number 0 with this device. The virual circuit number will also show up when using the **debug channel packets** command.

```
Channel3/0: sending command for vc 0, CLAW path C700, device C0
```

debug channel packets

Use the **debug channel packets** EXEC command to display per-packet debugging output. The output reports information when a packet is received or a transmit is attempted. The **no** form of this command disables debugging output.

debug channel packets
no debug channel packets

Syntax Description

This command has no arguments or keywords.

Command Mode

EXEC

Usage Guidelines

The **debug channel packets** command displays all process-level Channel Interface Processor (CIP) packets for both outbound and and inbound packets. You will need to disable fast switching and autonomous switching to obtain debugging output. This command is useful for determining whether packets are received or transmitted correctly.

This command is valid for the Cisco 7000 series routers only.

Sample Display

Figure 2-21 shows sample **debug channel packets** output.

Figure 2-21 Sample Debug Channel Packets Output

```
Router# debug channel packets

Channel packets debugging is on
(Channel3/0)-out size = 104, vc = 0000, type = 0800, src 198.92.0.11, dst 198.92.1.58
(Channel3/0)-in  size = 48, vc = 0000, type = 0800, src 198.92.1.58, dst 198.92.15.197
(Channel3/0)-in  size = 48, vc = 0000, type = 0800, src 198.92.1.58, dst 198.92.15.197
(Channel3/0)-out size = 71, vc = 0000, type = 0800, src 198.92.15.197, dst 198.92.1.58
(Channel3/0)-in  size = 44, vc = 0000, type = 0800, src 198.92.1.58, dst 198.92.15.197
```

Table 2-10 provides explanations for individual lines of output from Figure 2-21.

Table 2-10 Channel Packets Field Descriptions

Field	Description
(Channel3/0)	The interface slot and port.
in / out	In is a packet from the mainframe to the router. Out is a packet from the router to the mainframe.
size =	The number of bytes in the packet, including internal overhead.
vc =	A value from 0–511 that maps to the **claw** interface configuration command. This information is from the MAC layer.

Field	Description
type =	The encapsulation type in the MAC layer. The value 0800 indicates an IP datagram.
src	The origin, or source, of the packet, as opposed to the previous hop address.
dst	The destination of the packet, as opposed to the next hop address.

debug clns esis events

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

debug clns esis events
no debug clns esis events

Syntax Description

This command has no arguments or keywords.

Command Mode

EXEC

Sample Display

Figure 2-22 shows sample **debug clns esis events** output.

Figure 2-22 Sample Debug CLNS ESIS Events Output

```
router# debug clns esis events

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

Explanations for individual lines of output from Figure 2-22 follow.

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

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

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

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

The following line indicates that the router sent an IS hello packet on the Ethernet0 interface to all ESs on the network. The router's NET address is 49.0001.AA00.6904.00, the hold time for this packet is 299 seconds, and the header length of this packet is 20 bytes.

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

debug clns esis packets

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

debug clns esis packets
no debug clns esis packets

Syntax Description

This command has no arguments or keywords.

Command Mode

EXEC

Sample Display

Figure 2-23 shows sample **debug clns esis packets** output.

Figure 2-23 Sample Debug CLNS ESIS Packets Output

```
router# debug clns esis packets

ES-IS: ISH sent to All ESs (Ethernet0): NET
47.0005.80ff.ef00.0000.0001.5940.1600.8906.4023.00, HT 299, HLEN 33
ES-IS: ISH sent to All ESs (Ethernet1): NET
47.0005.80ff.ef00.0000.0001.5940.1600.8906.4023.00, HT 299, HLEN 34
ES-IS: ISH from aa00.0400.6408 (Ethernet0), HT 299
ES-IS: ISH sent to All ESs (Tunnel0): NET
47.0005.80ff.ef00.0000.0001.5940.1600.O906.4023.00, HT 299, HLEN 34
IS-IS: ESH from 0000.0c00.bda8 (Ethernet0), HT 300
```

Explanations for individual lines of output from Figure 2-23 follow.

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

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

The following line indicates that the router has sent an IS hello packet on Ethernet1 to all ESs on the network. This hello packet indicates that the router's NET is 47.0005.80ff.ef00.0000.0001.5940.1600.8906.4023.00. The hold time for this packet is 299 seconds. The packet header is 33 bytes in length.

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

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

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

The following line indicates that the router has sent an IS hello packet on Tunnel0 to all ESs on the network. This hello packet indicates that the router's NET is 47.0005.80ff.ef00.0000.0001.5940.1600.8906.4023.00. The hold time for this packet is 299 seconds. The packet header is 33 bytes in length.

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

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

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

debug clns events

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

debug clns events
no debug clns events

Syntax Description

This command has no arguments or keywords.

Command Mode

EXEC

Sample Display

Figure 2-24 shows sample **debug clns events** output.

Figure 2-24 Sample Debug CLNS Events Output

```
router# debug clns events

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

Explanations for individual lines of output from Figure 2-24 follow.

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

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

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

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

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

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

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

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

debug clns igrp packets

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

debug clns igrp packets
no debug clns igrp packets

Syntax Description

This command has no arguments or keywords.

Command Mode

EXEC

Sample Display

Figure 2-25 shows sample **debug clns igrp packets** output.

Figure 2-25 Sample Debug CLNS IGRP Packets Output

```
router# debug clns igrp packets

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

Explanations for individual lines of output from Figure 2-25 follow.

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

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

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

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

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

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

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

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

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

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

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

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

debug clns packet

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

debug clns packet
no debug clns packet

Syntax Description

This command has no arguments or keywords.

Command Mode

EXEC

Sample Display

Figure 2-26 shows sample **debug clns packet** output.

Figure 2-26 Sample Debug CLNS Packet Output

```
router# debug clns packet

CLNS: Forwarding packet size 157
      from 47.0023.0001.0000.0000.0003.0001.1920.3614.3002.00 STUPI-RBS
      to 47.0005.80ff.ef00.0000.0001.5940.1600.8906.4017.00
      via 1600.8906.4017 (Ethernet0 0000.0c00.bda8)
CLNS: Echo PDU received on Ethernet0 from 4
7.0005.80ff.ef00.0000.0001.5940.1600.8906.4017.00!
CLNS: Sending from 47.0005.80ff.ef00.0000.0001.5940.1600.8906.4023.00 to
47.0005.80ff.ef00.0000.0001.5940.1600.8906.4017.00
      via 1600.8906.4017 (Ethernet0 0000.0c00.bda8)
```

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

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

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

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

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

debug clns routing

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

debug clns routing
no debug clns routing

Syntax Description

This command has no arguments or keywords.

Command Mode

EXEC

Sample Display

Figure 2-27 shows sample **debug clns routing** output.

Figure 2-27 Sample Debug CLNS Routing Output

```
router# debug clns routing

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

Explanations for individual lines of output from Figure 2-27 follow.

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

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

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

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

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

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

debug compress

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

debug compress
no debug compress

Syntax Description

This command has no arguments or keywords.

Command Mode

EXEC

Sample Display

Figure 2-28 shows sample **debug compress** output.

Figure 2-28 Sample Debug Compress Output

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

Table 2-11 describes significant fields shown in Figure 2-28.

Table 2-11 Debug Compress Field Descriptions

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

debug decnet adj

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

debug decnet adj
no debug decnet adj

Syntax Description

This command has no arguments or keywords.

Command Mode

EXEC

Sample Display

Figure 2-29 shows sample **debug decnet adj** output.

Figure 2-29 Sample Debug DECnet Adj Output

```
router# debug decnet adj
DECnet adjacencies debugging is on
router#
DNET-ADJ: Level 1 hello from 1.3
DNET-ADJ: sending hellos
DNET-ADJ: Sending hellos to all routers on interface Ethernet0, blksize 1498
DNET-ADJ: Level 1 hello from 1.3
DNET-ADJ: 1.5 adjacency initializing
DNET-ADJ: sending triggered hellos
DNET-ADJ: Sending hellos to all routers on interface Ethernet0, blksize 1498
DNET-ADJ: Level 1 hello from 1.3
DNET-ADJ: 1.5 adjacency up
DNET-ADJ: Level 1 hello from 1.5
DNET-ADJ: 1.5 adjacency down, listener timeout
```

Explanations for representative lines of output in Figure 2-29 follow.

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

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

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

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

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

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

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

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

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

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

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

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

debug decnet connects

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

debug decnet connects
no debug decnet connects

Syntax Description

This command has no arguments or keywords.

Command Mode

EXEC

Usage Guidelines

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

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

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

Sample Display

Figure 2-30 shows sample **debug decnet connects** output.

Figure 2-30 Sample Debug DECnet Connects Output

```
router# debug decnet connects

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

Table 2-12 describes significant fields shown in Figure 2-30.

Table 2-12 Debug DECnet Connects Field Descriptions

Field	Description
DNET-CON:	Indicates that this is a **debug decnet connects** packet
list 300 item #2 matched	Indicates that a packet matched the second item in access list 300
src = 19.403	Indicates the source DECnet address for the packet
dst = 19.309	Indicates the destination DECnet address for the packet
on Ethernet0:	Indicates the router interface on which the access list filtering the packet was applied
permitted	Indicates that the access list permitted the packet

Field	Description
srcname = “RICK”	Indicates the originator user of the packet
srcuic = [0,017]	Indicates the source UIC of the packet
dstobj = 42	Indicates that DECnet object 42 is the destination
id=“USER”	Indicates the access user

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

debug decnet events

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

debug decnet events
no debug decnet events

Syntax Description

This command has no arguments or keywords.

Command Mode

EXEC

Sample Display

Figure 2-31 shows sample **debug decnet events** output.

Figure 2-31 Sample Debug DECnet Events Output

```
router# debug decnet events

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

Explanations for representative lines of output in Figure 2-31 follow.

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

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

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

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

debug decnet packet

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

debug decnet packet
no debug decnet packet

Syntax Description

This command has no arguments or keywords.

Command Mode

EXEC

Sample Display

Figure 2-32 shows sample **debug decnet packet** output.

Figure 2-32 Sample Debug DECnet Packet Output

```
router# debug decnet packet

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

Explanations for individual lines of output from Figure 2-32 follow.

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

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

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

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

debug decnet routing

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

debug decnet routing
no debug decnet routing

Syntax Description

This command has no arguments or keywords.

Command Mode

EXEC

Sample Display

Figure 2-33 shows sample **debug decnet routing** output.

Figure 2-33 Sample Debug DECnet Routing Output

```
router# debug decnet routing

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

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

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

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

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

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

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

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

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

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

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

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

debug dialer

Use the **debug dialer** 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.

debug dialer
no debug dialer

Syntax Description

This command has no arguments or keywords.

Command Mode

EXEC

Usage Guidelines

Table 2-13 describes the error messages that the **debug dialer** command can generate for a serial interface being used as a V.25bis dialer for dial-on-demand routing (DDR).

Table 2-13 Debug Dialer Message Descriptions for DDR

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

When DDR is enabled on the interface, information concerning the cause of any calls (called Dialing cause) may be displayed.

The following line of output for an IP packet lists the name of the DDR interface and the source and destination addresses of the packet:

```
Dialing cause: Serial0: ip (s=131.108.1.111 d=131.108.2.22)
```

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

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

debug 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.

debug dspu activation [*name*]
no debug dspu activation [*name*]

Syntax Description

name	(Optional) A host or PU name designation.

Command Mode

EXEC

Usage Guidelines

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

Sample Display

Figure 2-34 shows sample **debug dspu activation** output. Not all intermediate numbers are shown for the "activated" and "deactivated" logical unit (LU) address ranges.

Figure 2-34 Sample Debug DSPU Activation Output

```
router# debug dspu activation
DSPU: LS HOST3745 connected
DSPU: PU HOST3745 activated
DSPU: LU HOST3745-2 activated
DSPU: LU HOST3745-3 activated
. . .
DSPU: LU HOST3745-253 activated
DSPU: LU HOST3745-254 activated

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

Table 2-14 describes significant fields in the output shown in Figure 2-34.

Table 2-14 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 colon.
connected activated disconnected deactivated	Event that occured 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.

debug dspu packet [*name*]
no debug dspu packet [*name*]

Syntax Description

name	(Optional) A host or PU name designation.

Command Mode

EXEC

Usage Guidelines

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

Sample Display

Figure 2-35 shows sample **debug dspu packet** output.

Figure 2-35 Sample Debug DSPU Packet Output

```
router# debug dspu packet

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

Table 2-15 describes significant fields in the output shown in Figure 2-35.

Table 2-15 Debug DSPU Packet Field Descriptions

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

Related Commands

debug dspu activation
debug dspu state
debug dspu trace

debug dspu state

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

debug dspu state [*name*]
no debug dspu state [*name*]

Syntax Description

name	(Optional) A host or PU name designation.

Command Mode

EXEC

Usage Guidelines

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

Sample Display

Figure 2-36 shows sample **debug dspu state** output. Not all intermediate numbers are shown for the "activated" and "deactivated" logical unit (LU) address ranges.

Figure 2-36 Sample Debug DSPU State Output

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

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

Table 2-15 describes significant fields in the output shown in Figure 2-36.

Table 2-16 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.
HOST3745-253	Host name or PU name and LU address.
input=*input*,	The input received by the FSM.
previous-state, -> current-state	The previous state and current new state as seen by the FSM.

Related Commands

debug dspu activation
debug dspu packet
debug dspu trace

debug dspu trace

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

debug dspu trace [*name*]
no debug dspu trace [*name*]

Syntax Description

name	(Optional) A host or PU name designation.

Command Mode

EXEC

Usage Guidelines

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

Sample Display

Figure 2-37 shows sample **debug dspu trace** output.

Figure 2-37 Sample Debug DSPU Trace Output

```
router# debug dspu trace

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

Table 2-17 describes significant fields in the output shown in Figure 2-37.

Table 2-17 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.
in:*input* s:*state* ->(*new-state*, *action*)	String describing the following: *input* - LU FSM input *state* - Current FSM state *new-state* - New FSM state *action* - FSM action
`input=`*input* `->` `(`*new-state*`,`*action*`)`	String describing the following: *input* - PU or LS FSM input *new-state* - New PU or LS FSM state *action* - PU or LS FSM action

Related Commands

debug dspu activation
debug dspu packet
debug dspu state

debug eigrp fsm

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

debug eigrp fsm
no debug eigrp fsm

Syntax Description

This command has no arguments or keywords.

Command Mode

EXEC

Usage Guidelines

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

Sample Display

Figure 2-38 shows sample **debug eigrp fsm** output.

Figure 2-38 Sample Debug EIGRP FSM Output

```
router# debug eigrp fsm

DUAL: dual_rcvupdate(): 198.93.166.0 255.255.255.0 via 0.0.0.0 metric 750080/0
DUAL: Find FS for dest 198.93.166.0 255.255.255.0. FD is 4294967295, RD is 42949
67295 found
DUAL: RT installed 198.93.166.0 255.255.255.0 via 0.0.0.0
DUAL: dual_rcvupdate(): 192.168.4.0 255.255.255.0 via 0.0.0.0 metric 4294967295/
4294967295
DUAL: Find FS for dest 192.168.4.0 255.255.255.0. FD is 2249216, RD is 2249216
DUAL:   0.0.0.0 metric 4294967295/4294967295not found Dmin is 4294967295
DUAL: Dest 192.168.4.0 255.255.255.0 not entering active state.
DUAL: Removing dest 192.168.4.0 255.255.255.0, nexthop 0.0.0.0
DUAL: No routes.  Flushing dest 192.168.4.0 255.255.255.0
```

Explanations for individual lines of output from Figure 2-38 follow.

In the first line of Figure 2-38, DUAL stands for Diffusing Update ALgorithm. It is the basic mechanism within Enhanced IGRP that makes the routing decisions.The next three fields are the Internet address and mask of the destination network and the address through which the update was received. The metric field shows the metric stored in the routing table and the metric advertised by the neighbor sending the information. "Metric ... inaccessible" usually means that the neighbor router no longer has a route to the destination, or the destination is in holddown.

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

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

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

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

```
DUAL: RT installed 198.93.166.0 255.255.255.0 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.

debug eigrp packet
no debug eigrp packet

Syntax Description

This command has no arguments or keywords.

Command Mode

EXEC

Usage Guidelines

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

Sample Display

Figure 2-39 shows sample **debug eigrp packet** output.

Figure 2-39 Sample Debug EIGRP Packet Output

```
router# debug eigrp packet

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

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

Table 2-18 describes significant fields in the output shown in Figure 2-39.

Table 2-18 Debug EIGRP Packet Field Descriptions

Field	Description
EIGRP:	An Enhanced IGRP packet.
AS *n*	Autonomous System number.
Flags *n*x*n*	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 contitionally-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.
HELLO	The hello packets are the neighbor discovery packets. They are used to determine if 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 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.

debug frame-relay
no debug frame-relay

Syntax Description

This command has no arguments or keywords.

Command Mode

EXEC

Usage Guidelines

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

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

Sample Display

Figure 2-40 shows sample **debug frame-relay** output.

Figure 2-40 Sample Debug Frame-Relay Output

```
router# debug frame-relay

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

Table 2-19 describes significant fields shown in Figure 2-40.

Table 2-19 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.

Field	Description
pkt type 0x809B	Indicates the packet type code. Possible supported signaling message codes follow: 0x308—Signaling message; valid only with a DLCI of 0. 0x309—LMI message; valid only with a DLCI of 1023 Possible supported Ethernet type codes follow: 0x0201—IP on 3MB net 0x0201—Xerox ARP on 10MB nets 0xCC—RFC 1294 (only for IP) 0x0600—XNS 0x0800—IP on 10 MB net 0x0806—IP ARP 0x0808—Frame Relay ARP 0x0BAD—VINES IP 0x0BAE—VINES loopback protocol 0x0BAF—VINES Echo 0x6001—DEC MOP booting protocol 0x6002—DEC MOP console protocol 0x6003—DECnet Phase IV on Ethernet 0x6004—DEC LAT on Ethernet 0x8005—HP Probe 0x8035—RARP 0x8038—DEC spanning tree 0x809b—Apple EtherTalk 0x80f3—AppleTalk ARP 0x8019—Apollo domain 0x80C4—VINES IP 0x80C5— VINES ECHO 0x8137—IPX 0x9000—Ethernet loopback packet IP

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

debug frame-relay events

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

debug frame-relay events
no debug frame-relay events

Syntax Description

This command has no arguments or keywords.

Command Mode

EXEC

Usage Guidelines

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

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

Sample Display

Figure 2-41 shows sample **debug frame-relay events** output.

Figure 2-41 Sample Debug Frame-Relay Events Output

```
router# debug frame-relay events

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

As Figure 2-41 shows, **debug frame-relay events** returns one specific message type. The first line, for example, indicates that IP address 131.108.170.26 sent a Frame Relay ARP reply; this packet was received as input on the Serial2 interface. The last field (126) is the data link connection identifier (DLCI) to use when communicating with the responding router.

debug frame-relay lmi

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

debug frame-relay lmi
no debug frame-relay lmi

Syntax Description

This command has no arguments or keywords.

Command Mode

EXEC

Usage Guidelines

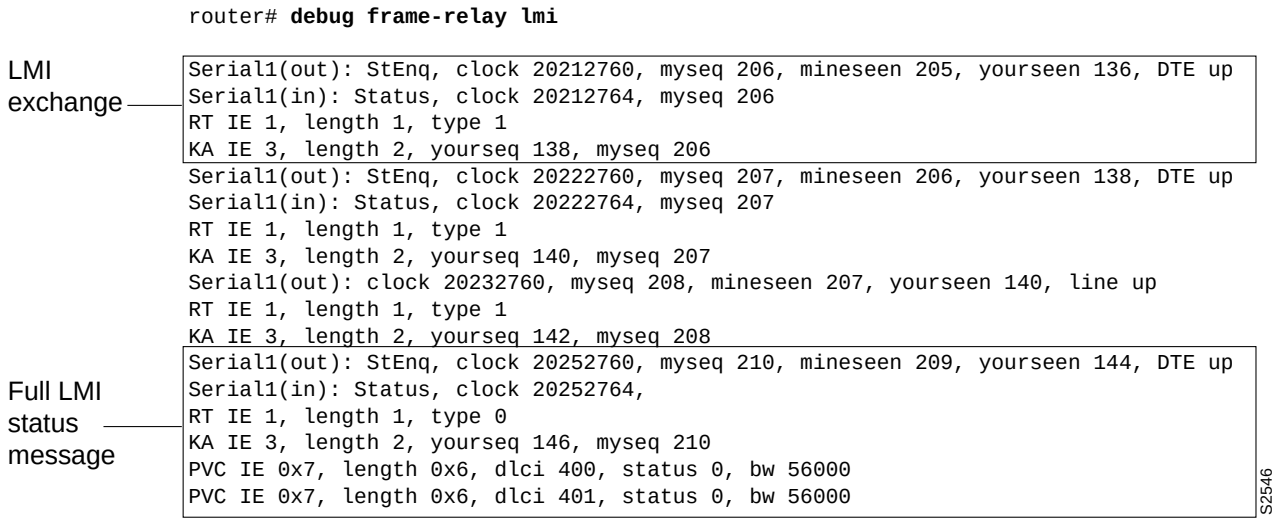
You can use this command to determine whether the router and the Frame Relay switch are sending and receiving LMI packets properly.

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

Sample Display

Figure 2-42 shows sample **debug frame-relay lmi** output.

Figure 2-42 Sample Debug Frame-Relay LMI Output



In Figure 2-42, the first four lines describe an LMI exchange. The first line describes the LMI request the router has sent to the switch. The second line describes the LMI reply the router has received from the switch. The third and fourth lines describe the response to this request from the switch. This LMI exchange is followed by two similar LMI exchanges. The last six lines in Figure 2-42 consist of a full LMI status message that includes a description of the router's two permanent virtual circuits (PVCs).

Table 2-20 describes significant fields in the first line of the **debug frame-relay lmi** output shown in Figure 2-42.

Table 2-20 Debug Frame-Relay LMI Field Descriptions—Part 1

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

Table 2-21 describes significant fields in the third and fourth lines of **debug frame-relay lmi** output shown in Figure 2-42.

Table 2-21 Debug Frame-Relay LMI Field Descriptions—Part 2

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

Table 2-22 describes significant fields in the last line of **debug frame-relay lmi** output shown in Figure 2-42.

Table 2-22 Debug Frame-Relay LMI Field Descriptions—Part 3

Field	Description
PVC IE 0x7	Value of the permanent virtual circuit information element type.
length 0x6	Length of the PVC IE (in bytes).
dlci 401	DLCI decimal value for this PVC.
status 0	Status value. Possible values include the following: 0x00—Added/inactive 0x02—Added/active 0x04—Deleted 0x08—New/inactive 0x0a—New/active
bw 56000	CIR (committed information rate), in decimal, for the DLCI.

debug frame-relay packets

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

debug frame-relay packets
no debug frame-relay packets

Syntax Description

This command has no arguments or keywords.

Command Mode

EXEC

Usage Guidelines

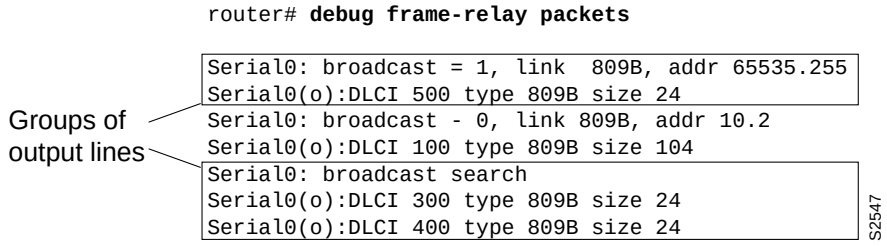
This command helps you analyze the packets that are sent on a Frame Relay interface. Because the **debug frame-relay packets** command generates large amounts of output, only use it when traffic on the Frame Relay network is less than 25 packets per second.

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

Sample Display

Figure 2-43 shows sample **debug frame-relay packets** output.

Figure 2-43 Sample Debug Frame-Relay Packets Output



As Figure 2-43 shows, **debug frame-relay packets** output consists of groups of output lines; each group describes a Frame Relay packet that has been sent. The number of lines in the group can vary, depending on the number of data link connection identifiers (DLCIs) on which the packet was sent. For example, the first two pairs of output lines describe two different packets, both of which were sent out on a single DLCI. The last three lines in Figure 2-43 describe a single Frame Relay packet that was sent out on two DLCIs.

Table 2-23 describes significant fields shown in the first pair of output lines in Figure 2-43.

Table 2-23 Debug Frame-Relay Packets Field Descriptions

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

Explanations for other lines of output shown in Figure 2-43 follow:

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

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

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

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

The following lines do not appear in Figure 2-43. They describe a Frame Relay packet sent to a true broadcast address.

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

debug ip dvmrp

Use the **debug ip dvmrp** EXEC command to display information on Distance Vector Multiprotocol Routing Protocol (DVMRP) packets received and transmitted. The **no** form of this command disables debugging output.

debug ip dvmrp [**detail**]
no debug ip dvmrp

Syntax Description

detail	(Optional) Enables a more detailed level of output and displays packet contents.

Command Mode

EXEC

Usage Guidelines

Use the **debug ip dvmrp detail** command with care. This command generates a great deal of output and can interrupt other activity on the router when it is invoked.

Sample Display

Figure 2-44 shows sample **debug ip dvmrp** output.

Figure 2-44 Sample Debug IP DVMRP Output

```
router# debug ip dvmrp
DVMRP: Received Report on Ethernet0 from 131.119.244.10
DVMRP: Received Report on Ethernet0 from 131.119.244.11
DVMRP: Building Report for Ethernet0 224.0.0.4
DVMRP: Send Report on Ethernet0 to 224.0.0.4
DVMRP: Sending IGMP Reports for known groups on Ethernet0
DVMRP: Received Report on Ethernet0 from 131.119.244.10
DVMRP: Received Report on Tunnel0 from 198.104.199.254
DVMRP: Received Report on Tunnel0 from 198.104.199.254
DVMRP: Received Report on Tunnel0 from 198.104.199.254
DVMRP: Received Report on Tunnel0 from 198.104.199.254
DVMRP: Received Report on Tunnel0 from 198.104.199.254
DVMRP: Received Report on Tunnel0 from 198.104.199.254
DVMRP: Building Report for Tunnel0 224.0.0.4
DVMRP: Send Report on Tunnel0 to 198.104.199.254
DVMRP: Send Report on Tunnel0 to 198.104.199.254
DVMRP: Send Report on Tunnel0 to 198.104.199.254
DVMRP: Send Report on Tunnel0 to 198.104.199.254
DVMRP: Radix tree walk suspension
DVMRP: Send Report on Tunnel0 to 198.104.199.254
```

Explanations for individual lines of output from Figure 2-44 follow.

The following lines show that the router received DVMRP routing information and placed it in the mroute table:

```
DVMRP: Received Report on Ethernet0 from 131.119.244.10
DVMRP: Received Report on Ethernet0 from 131.119.244.11
```

The following lines show that the router is creating a report to send to other DVMRP router:

```
DVMRP: Building Report for Ethernet0 224.0.0.4
DVMRP: Send Report on Ethernet0 to 224.0.0.4
```

Table 2-24 provides a list of internet multicast addresses supported for host IP implementations.

Table 2-24 Internet Multicast Addresses

Address	Description	RFC
224.0.0.0	Base address (Reserved)	RFC 1112
224.0.0.1	All systems on this subnet	RFC 1112
224.0.0.2	All routers on this subnet	
224.0.0.3	Unassigned	
224.0.0.4	DVMRP routers	RFC 1075
224.0.0.5	OSPFIGP all routers	RFC 1583

The following lines show that a protocol update report has been sent to all known multicast groups. Hosts use IGMP reports to communiate with routers and to request to join a multicast group. In this case, the router is sending an IGMP report for every known group to the host, which is running mrouted. The host the responds as though the router was a host on the LAN segment that wants to receive multicast packets for the group.

```
DVMRP: Sending IGMP Reports for known groups on Ethernet0
```

Figure 2-45 shows sample **debug ip dvmrp detail** output.

Figure 2-45 Sample Debug IP DVMRP Detail Output

```
router# debug ip dvmrp detail

DVMRP: Sending IGMP Reports for known groups on Ethernet0
DVMRP: Advertise group 224.2.224.2 on Ethernet0
DVMRP: Advertise group 224.2.193.34 on Ethernet0
DVMRP: Advertise group 224.2.231.6 on Ethernet0
DVMRP: Received Report on Tunnel0 from 198.104.199.254
DVMRP: Origin 150.166.53.0/24, metric 13, distance 0
DVMRP: Origin 150.166.54.0/24, metric 13, distance 0
DVMRP: Origin 150.166.55.0/24, metric 13, distance 0
DVMRP: Origin 150.166.56.0/24, metric 13, distance 0
DVMRP: Origin 150.166.92.0/24, metric 12, distance 0
DVMRP: Origin 150.166.100.0/24, metric 12, distance 0
DVMRP: Origin 150.166.101.0/24, metric 12, distance 0
DVMRP: Origin 150.166.142.0/24, metric 8, distance 0
DVMRP: Origin 150.166.200.0/24, metric 12, distance 0
DVMRP: Origin 150.166.237.0/24, metric 12, distance 0
DVMRP: Origin 150.203.5.0/24, metric 8, distance 0
```

Explanations for individual lines of output from Figure 2-45 follow.

The following lines show that this group is available to the DVMRP router. The mrouted process on the host will forward the S,G information for theis group through the DVMRP cloud so other members will know this S,G is available.

```
DVMRP: Advertise group 224.2.224.2 on Ethernet0
```

The following lines show the DVMRP route information:

```
DVMRP: Origin 150.166.53.0/24, metric 13, distance 0
DVMRP: Origin 150.166.54.0/24, metric 13, distance 0
```

Metric is the number of hops the route has covered. Distance is the administrative distance.

debug ip eigrp

Use the **debug ip eigrp** EXEC command to display information on Enhanced IGRP protocol packets. The **no** form of this command disables debugging output.

debug ip eigrp
no debug ip eigrp

Syntax Description

This command has no arguments or keywords.

Command Mode

EXEC

Usage Guidelines

This command helps you analyze the packets that are sent and received on an interface. Because the **debug ip eigrp** command generates large amounts of output, only use it when traffic on the network is light.

Sample Display

Figure 2-46 shows sample **debug ip eigrp** output.

Figure 2-46 Sample Debug IP EIGRP Output

```
router# debug ip eigrp

IP-EIGRP: Processing incoming UPDATE packet
IP-EIGRP: Ext 198.135.3.0 255.255.255.0 M 386560 - 256000 130560 SM 360960 - 256
000 104960
IP-EIGRP: Ext 198.135.0.0 255.255.255.0 M 386560 - 256000 130560 SM 360960 - 256
000 104960
IP-EIGRP: Ext 198.135.3.0 255.255.255.0 M 386560 - 256000 130560 SM 360960 - 256
000 104960
IP-EIGRP: 198.92.43.0 255.255.255.0, - do advertise out Ethernet0/1
IP-EIGRP: Ext 198.92.43.0 255.255.255.0 metric 371200 - 256000 115200
IP-EIGRP: 192.135.246.0 255.255.255.0, - do advertise out Ethernet0/1
IP-EIGRP: Ext 192.135.246.0 255.255.255.0 metric 46310656 - 45714176 596480
IP-EIGRP: 198.92.40.0 255.255.255.0, - do advertise out Ethernet0/1
IP-EIGRP: Ext 198.92.40.0 255.255.255.0 metric 2272256 - 1657856 614400
IP-EIGRP: 192.135.245.0 255.255.255.0, - do advertise out Ethernet0/1
IP-EIGRP: Ext 192.135.245.0 255.255.255.0 metric 40622080 - 40000000 622080
IP-EIGRP: 192.135.244.0 255.255.255.0, - do advertise out Ethernet0/1
```

Table 2-25 describes significant fields in the debug messages shown in Figure 2-46.

Table 2-25 Debug IP EIGRP Field Descriptions

Field	Description
IP-EIGRP:	Indicates that this is an IP Enhanced IGRP packet.
Ext	Indicates the following address is an external destination rather than an internal destination, which would be labeled as Int.
M	Shows the computed metric, which includes SM and the cost between this router and the neighbor. The first number is the composite metric. The next two numbers are the inverse bandwidth and the delay, respectively.
SM	Shows the metric as reported by the neighbor.

debug ip icmp

Use the **debug ip icmp** EXEC command to display information on Internal Control Message Protocol (ICMP) transactions. The **no** form of this command disables debugging output.

debug ip icmp
no debug ip icmp

Syntax Description

This command has no arguments or keywords.

Command Mode

EXEC

Usage Guidelines

This command helps you determine whether the router is sending or receiving ICMP messages. Use it, for example, when you are troubleshooting an end-to-end connection problem.

Sample Display

Figure 2-47 shows sample **debug ip icmp** output.

Figure 2-47 Sample Debug IP ICMP Output

```
router# debug ip icmp

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

Table 2-26 describes significant fields in the first line of **debug ip icmp** output shown in Figure 2-47.

Table 2-26 Debug IP ICMP Field Descriptions—Part 1

Field	Description
ICMP:	Indication that this message describes an ICMP packet.
rcvd type 3	The type field can be one of the following: 0—Echo Reply 3—Destination Unreachable 4—Source Quench 5—Redirect 8—Echo 9—Router Discovery Protocol Advertisement 10—Router Discovery Protocol Solicitations 11—Time Exceeded 12—Parameter Problem 13—Timestamp 14—Timestamp Reply 15—Information Request 16—Information Reply 17—Mask Request 18—Mask Reply
code 1	This field is a code. The meaning of the code depends upon the type field value: Echo and Echo Reply—The code field is always zero. Destination Unreachable—The code field can have the following values: 0—Network unreachable 1—Host unreachable 2—Protocol unreachable 3—Port unreachable 4—Fragmentation needed and DF bit set 5—Source route failed Source Quench—The code field is always 0. Redirect—The code field can have the following values: 0—Redirect datagrams for the network 1—Redirect datagrams for the host 2—Redirect datagrams for the command mode of service and network 3—Redirect datagrams for the command mode of service and host Router Discovery Protocol Advertisements and Solicitations—The code field is always zero.

Field	Description
code 1 (continued)	Time Exceeded—The code field can have the following values: 0—Time to live exceeded in transit 1—Fragment reassembly time exceeded Parameter Problem—The code field can have the following values: 0—General problem 1—Option is missing 2—Option missing, no room to add Timestamp and Timestamp Reply—The code field is always zero. Information Request and Information Reply—The code field is always zero. Mask Request and Mask Reply—The code field is always zero.
from 128.95.192.4	Source address of the ICMP packet.

Table 2-27 describes significant fields in the second line of **debug ip icmp** output in Figure 2-47.

Table 2-27 Debug IP ICMP Field Descriptions—Part 2

Field	Description
ICMP:	Indication that this message describes an ICMP packet
src 36.56.0.202	The address of the sender of the echo
dst 131.108.16.1	The address of the receiving router
echo reply	Indication the router received an echo reply

Other messages that the **debug ip icmp** command can generate follow.

When an IP router or host sends out an ICMP mask request, the following message is generated when the router sends a mask reply:

```
ICMP: sending mask reply (255.255.255.0) to 160.89.80.23 via Ethernet0
```

The following two lines are examples of the two forms of this message. The first form is generated when a mask reply comes in after the router sends out a mask request. The second form occurs when the router receives a mask reply with a nonmatching sequence and ID. See Appendix I of RFC 950, "Internet Standard Subnetting Procedures," for details.

```
ICMP: mask reply 255.255.255.0 from 160.89.80.31
ICMP: unexpected mask reply 255.255.255.0 from 160.89.80.32
```

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

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

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

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

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

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

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

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

Depending on the code received (as Table 2-26 describes), any of the unreachable messages can have any of the following "strings" instead of the "host" string in the message:

```
net
protocol
port
frag. needed and DF set
source route failed
prohibited
```

The following message is displayed when the TTL in the IP header reaches zero and a time exceed ICMP message is sent. The fields are self-explanatory.

```
ICMP: time exceeded (time to live) send to 128.95.1.4 (dest was 131.108.1.111)
```

The following message is generated when parameters in the IP header are corrupted in some way and the parameter problem ICMP message is sent. The fields are self-explanatory.

```
ICMP: parameter problem sent to 128.121.1.50 (dest was 131.108.1.111)
```

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

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

Note For more information about the fields in **debug ip icmp** output, see RFC-792, "Internet Control Message Protocol"; Appendix I of RFC-950, "Internet Standard Subnetting Procedure"; and RFC-1256, "ICMP Router Discovery Messages."

---

debug ip igmp

Use the **debug ip igmp** EXEC command to display Internet Group Management Protocol (IGMP) packets received and transmitted, as well as IGMP-host related events. The **no** form of this command disables debugging output.

debug ip igmp
no debug ip igmp

Syntax Description

This command has no arguments or keywords.

Command Mode

EXEC

Usage Notes

This command helps discover whether the IGMP processes are functioning. In general, if IGMP is not working, the router process never discovers that there is another host on the network that is configured to receive multicast packets. In dense mode this means the packets will be delivered intermittently (a few every 3 minutes). In sparse mode they will never be delivered.

Use this command in conjunction with **debug ip pim** and **debug ip mrouting** to observe additional multicast activity and to see what is happening the the multicast routing process, or why packets are forwarded out of particular interfaces.

Sample Display

Figure 2-48 shows sample **debug ip igmp** output.

Figure 2-48 Sample Debug IP IGMP Output

```
router# debug ip igmp

IGMP: Received Host-Query from 198.92.37.33 (Ethernet1)
IGMP: Received Host-Report from 198.92.37.192 (Ethernet1) for 224.0.255.1
IGMP: Received Host-Report from 198.92.37.57 (Ethernet1) for 224.2.127.255
IGMP: Received Host-Report from 198.92.37.33 (Ethernet1) for 225.2.2.2
```

Explanations for output from Figure 2-48 follow.

The messages displayed by the **debug ip igmp** command show query and report activity received from other routers and multicast group addresses.

Related Commands

debug ip pim
debug ip mrouting

debug ip igrp events

Use the **debug ip igrp events** EXEC command to display summary information on Interior Gateway Routing Protocol (IGRP) routing messages that indicates the source and destination of each update, as well as the number of routes in each update. Messages are not generated for each route. The **no** form of this command disables debugging output.

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

Syntax Description

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

Command Mode

EXEC

Usage Guidelines

If the IP address of an IGRP neighbor is specified, the resulting **debug ip igrp events** output includes messages describing updates from that neighbor and updates that the router broadcasts toward that neighbor.

This command is particularly useful when there are many networks in your routing table. In this case, using **debug ip igrp transaction** could flood the console and make the router unusable. Use **debug ip igrp events** instead to display summary routing information.

Sample Display

Figure 2-49 shows sample **debug ip igrp events** output.

Figure 2-49 Sample Debug IP IGRP Events Output

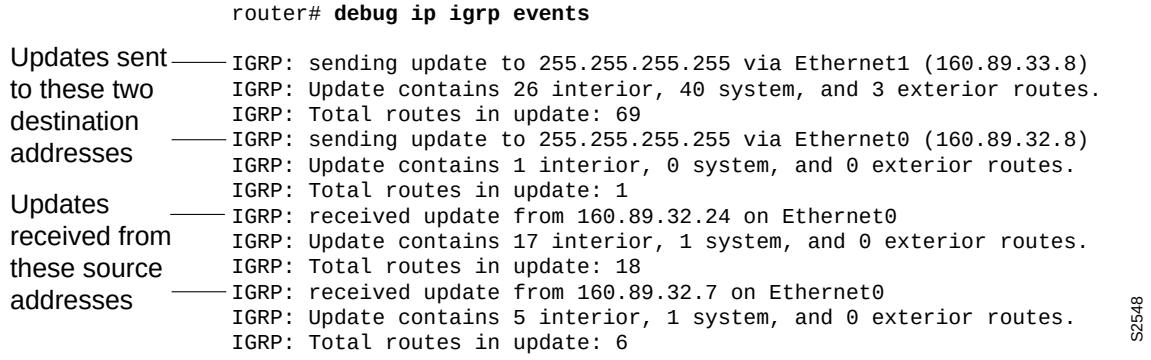


Figure 2-49 shows that the router has sent two updates to the broadcast address 255.255.255.255. The router also received two updates. Three lines of output describe each of these updates. Explanations for representative lines of output from Figure 2-49 follow.

The first line indicates whether the router sent or received the update packet, the source or destination address, and the interface through which the update was sent or received. If the update was sent, the IP address assigned to this interface is shown (in parentheses).

```
IGRP: sending update to 255.255.255.255 via Ethernet1 (160.89.33.8)
```

The second line summarizes the number and types of routes described in the update:

```
IGRP: Update contains 26 interior, 40 system, and 3 exterior routes.
```

The third line indicates the total number of routes described in the update.

```
IGRP: Total routes in update: 69
```

debug ip igrp transaction

Use the **debug ip igrp transaction** EXEC command to display transaction information on Interior Gateway Routing Protocol (IGRP) routing transactions. The **no** form of this command disables debugging output.

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

Syntax Description

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

Command Mode

EXEC

Usage Guidelines

If the IP address of an IGRP neighbor is specified, the resulting **debug ip igrp transaction** output includes messages describing updates from that neighbor and updates that the router broadcasts toward that neighbor.

When there are many networks in your routing table, **debug ip igrp transaction** can flood the console and make the router unusable. In this case, use **debug ip igrp events** instead to display summary routing information.

Sample Display

Figure 2-50 shows sample **debug ip igrp transaction** output.

Figure 2-50 Sample Debug IP IGRP Transaction Output

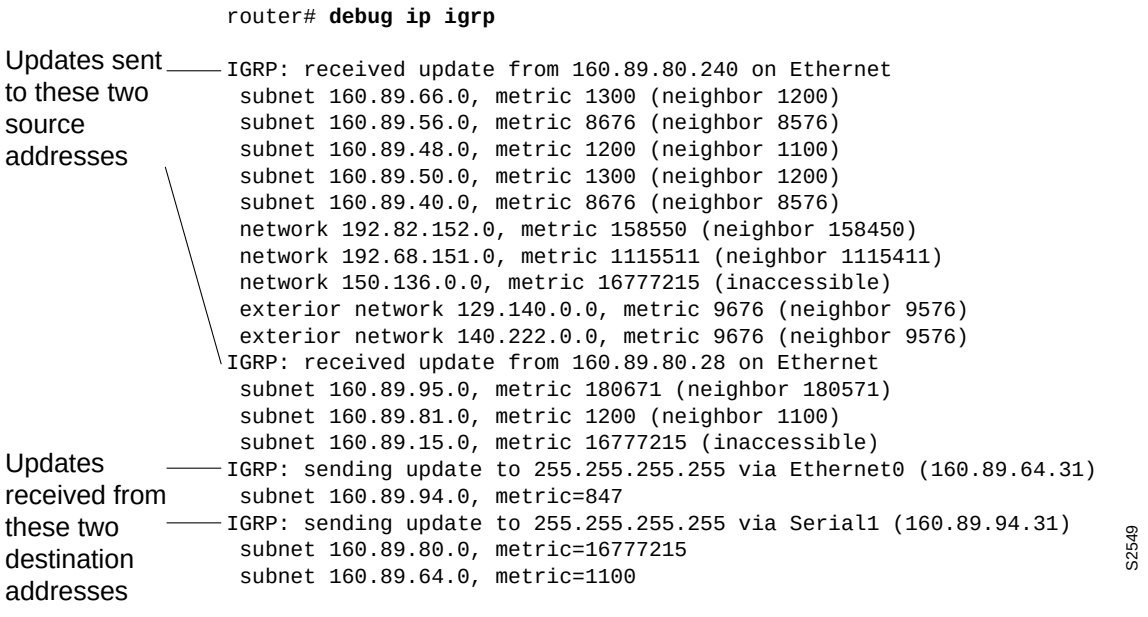


Figure 2-50 shows that the router being debugged has received updates from two other routers on the network. The router at source address 160.89.80.240 sent information about ten destinations in the update; the router at source address 160.89.80.28 sent information about three destinations in its update. The router being debugged also sent updates—in both cases to the broadcast address 255.255.255.255 as the destination address.

The first line in Figure 2-50 is self-explanatory.

On the second line in Figure 2-50, the first field refers to the type of destination information: "subnet" (interior), "network" (system), or "exterior" (exterior). The second field is the Internet address of the destination network. The third field is the metric stored in the routing table and the metric advertised by the neighbor sending the information. "Metric ... inaccessible" usually means that the neighbor router has put the destination in holddown.

The entries in Figure 2-50 show that the router is sending updates that are similar, except that the numbers in parentheses are the source addresses used in the IP header. A metric of 16777215 is inaccessible.

Other examples of output that the **debug ip igrp transaction** command can produce follow.

The following entry indicates that the routing table was updated and shows the new edition number (97 in this case) to be used in the next IGRP update:

```
IGRP: edition is now 97
```

Entries such as the following occur on startup or when some event occurs such as an interface transitioning or a user manually clearing the routing table:

```
IGRP: broadcasting request on Ethernet0
IGRP: broadcasting request on Ethernet1
```

The following type of entry can result when routing updates become corrupted between sending and receiving routers:

```
IGRP: bad checksum from 160.89.64.43
```

An entry such as the following should never appear. If it does, the receiving router has a bug in the software or a problem with the hardware. In either case, contact your technical support representative.

```
IGRP: system 45 from 160.89.64.234, should be system 109
```

debug ip mpacket

Use the **debug ip mpacket** EXEC command to display only IP multicast packets received and transmitted.The **no** form of this command disables debugging output.

debug ip mpacket [*group*]
no debug ip mpacket [*group*]

Syntax Description

group	(Optional) Group name or address to monitor a single group's packet activity

Command Mode

EXEC

Usage Guidelines

This command displays information for multicast IP packets that are forwarded from this router. By using the optional *group*, you can limit the display to a specific multicast group.

Use this command with **debug ip packet** to observe additional packet information.

Note The **debug ip mpacket** command generates lots of messages. Use with care so that performance on the network is not affected by the debug message traffic.

Sample Display

Figure 2-51 shows sample **debug ip mpacket** output.

Figure 2-51 Sample Debug IP Mpacket Output

```
router# debug ip mpacket 224.2.0.1

IP: s=131.188.34.54 (Ethernet1), d=224.2.0.1 (Tunnel0), len 88, mforward
IP: s=131.188.34.54 (Ethernet1), d=224.2.0.1 (Tunnel0), len 88, mforward
IP: s=131.188.34.54 (Ethernet1), d=224.2.0.1 (Tunnel0), len 88, mforward
IP: s=140.162.3.27 (Ethernet1), d=224.2.0.1 (Tunnel0), len 68, mforward
```

Table 2-28 defines fields shown in Figure 2-51.

Table 2-28 Debug IP Mpacket Field Descriptions

Field	Description
IP	An IP packet.
s= *address*	The source address of the packet.
(Ethernet1)	The name of the interface that received the packet.
d= *address*	The multicast group address that is the destination for this packet.

Field	Description
(Tunnel0)	The outgoing interface for the packet.
len 88	The number of bytes in the packet. This value will vary depending on the application and the media.
mforward	The packet has been forwarded.
not RPF interface	The interface is not a reverse packet forwarding interface. (See **debug ip mrouting**.)
RPF lookup failed	The reverse packet forwarding lookup failed. (See **debug ip mrouting**.)

Related Commands

debug ip mrouting
debug ip packet

debug ip mrouting

Use the **debug ip mrouting** EXEC command to display changes to the IP multicast routing table.The **no** form of this command disables debugging output.

debug ip mrouting [*group*]
no debug ip mrouting [*group*]

Syntax Description

group	(Optional) Group name or address to monitor a single group's packet activity

Command Mode

EXEC

Usage Notes

This command tells when the router has made changes to the mroute table. Use the **debug ip pim** and **debug ip mrouting** commands at the same time to obtain additional multicast routing information. In addition, use the **debug ip igmp** command to see why an mroute message is being displayed.

This command generates a large amount of output. Use the optional *group* to limit the output to a single multicast group.

Sample Display

Figure 2-52 shows sample **debug ip mrouting** output.

Figure 2-52 Sample Debug IP Mrouting Output

```
router# debug ip mrouting 224.2.0.1
IP multicast routing debugging is on

MRT: Delete (13.0.0.0/8, 224.2.0.1)
MRT: Delete (128.3.0.0/16, 224.2.0.1)
MRT: Delete (128.6.0.0/16, 224.2.0.1)
MRT: Delete (128.9.0.0/16, 224.2.0.1)
MRT: Delete (128.16.0.0/16, 224.2.0.1)
MRT: Create (*, 224.2.0.1), if_input NULL
MRT: Create (198.92.15.0/24, 225.2.2.4), if_input Ethernet0, RPF nbr 131.108.61.15
MRT: Create (198.92.39.0/24, 225.2.2.4), if_input Ethernet1, RPF nbr 0.0.0.0
MRT: Create (13.0.0.0/8, 224.2.0.1), if_input Ethernet1, RPF nbr 0.0.0.0
MRT: Create (128.3.0.0/16, 224.2.0.1), if_input Ethernet1, RPF nbr 0.0.0.0
MRT: Create (128.6.0.0/16, 224.2.0.1), if_input Ethernet1, RPF nbr 0.0.0.0
MRT: Create (128.9.0.0/16, 224.2.0.1), if_input Ethernet1, RPF nbr 0.0.0.0
MRT: Create (128.16.0.0/16, 224.2.0.1), if_input Ethernet1, RPF nbr 0.0.0.0
```

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

The following lines show that multicast IP routes were deleted from the routing table:

```
MRT: Delete (13.0.0.0/8, 224.2.0.1)
MRT: Delete (128.3.0.0/16, 224.2.0.1)
MRT: Delete (128.6.0.0/16, 224.2.0.1)
```

The *,G entry in the following line is always null since it is a *,G. The *,G entries are generally created by receipt of an IGMP host-report from a group member on the directly connected lan or by a PIM join message (in sparse mode) which this router receives from a router that is sending joins toward the RP. This router will in turn, send a join toward the RP which creates the shared tree (or RP tree).

```
MRT: Create (*, 224.2.0.1), if_input NULL
```

The following lines are an example of creating an S,G entry that show a mpacket was received on E0. The second line shows a route being created for a source that is on a directly connected LAN. The RPF means “reverse path forwarding,” whereby the router looks up the source address of the multicast packet in the unicast routing table and asks which interface will be used to send a packet to that source.

```
MRT: Create (198.92.15.0/24, 225.2.2.4), if_input Ethernet0, RPF nbr 131.108.61.15
MRT: Create (198.92.39.0/24, 225.2.2.4), if_input Ethernet1, RPF nbr 0.0.0.0
```

The following lines show that multicast IP routes were added to the routing table. Note the 0.0.0.0 as the RPF, which means the route was created by a source that is directly connected to this router.

```
MRT: Create (128.9.0.0/16, 224.2.0.1), if_input Ethernet1, RPF nbr 0.0.0.0
MRT: Create (128.16.0.0/16, 224.2.0.1), if_input Ethernet1, RPF nbr 0.0.0.0
```

If the source is not directly connected, the nbr address shown in these lines will be the address of the router that forwarded the packet to this router.

The shortest path tree state maintained in routers consists of source (S), multicast address (G), outgoing interface (OIF), and incoming interface (IIF). The forwarding information is referred to as the multicast forwarding entry for (S,G).

An entry for a shared tree can match packets from any source for its associated group if the packets come through the proper incoming interface as determined by the RPF lookup. Such an entry is denoted as (*,G). A (*,G) entry keeps the same information a (S,G) entry keeps, except that it saves the rendezvous point (RP) address in place of the source address in sparse mode or 0.0.0.0 in dense mode.

Related Commands

debug ip pim
debug ip igmp

debug ip ospf events

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

debug ip ospf events
no debug ip ospf events

Syntax Description

This command has no arguments or keywords.

Command Mode

EXEC

Sample Display

Figure 2-53 shows sample **debug ip ospf events** output.

Figure 2-53 Sample Debug IP OSPF Events Output

```
router# debug ip ospf-events

OSPF:hello with invalid timers on interface Ethernet0
hello interval received 10  configured 10
net mask received 255.255.255.0  configured 255.255.255.0
dead interval received 40  configured 30
```

The **debug ip ospf events** output shown in Figure 2-53 might appear if any of the following occurs:

- The IP subnet masks for routers on the same network do not match.
- The OSPF hello interval for the router does not match that configured for a neighbor.
- The OSPF dead interval for the router does not match that configured for a neighbor.

If a router configured for OSPF routing is not seeing an OSPF neighbor on an attached network, do the following:

- Make sure that both routers have been configured with the same IP mask, OSPF hello interval, and OSPF dead interval.
- Make sure that both neighbors are part of the same area type.

In the following example line, the neighbor and this router are not part of a stub area (that is, one is a part of a transit area and the other is a part of a stub area, as explained in RFC 1247).

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

debug ip packet

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

debug ip packet [*access-list-number*]
no debug ip packet [*access-list-number*]

Syntax Description

access-list-number	(Optional) IP access list number that you can specify. If the datagram is not permitted by that access list, the related debugging output is suppressed.

Command Mode

EXEC

Usage Guidelines

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

IP debugging information includes packets received, generated, and forwarded. Fast-switched packets do not generate messages.

IPSO security transactions include messages that describe the cause of failure each time a datagram fails a security test in the system. This information is also sent to the sending host when the router configuration allows it.

Note Because the **debug ip packet** command generates a significant amount of output, use it only when traffic on the IP network is low, so other activity on the system is not adversely affected.

Sample Display

Figure 2-54 shows sample **debug ip packet** output.

Figure 2-54 Sample Debug IP Packet Output

```
router# debug ip packet

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

Figure 2-54 shows two types of messages that the **debug ip packet** command can produce; the first line of output describes an IP packet that the router forwards, and the third line of output describes a packet that is destined for the router. In the third line of output, "rcvd 2" indicates that the router decided to receive the packet.

Table 2-29 describes the fields shown in the first line of Figure 2-54.

Table 2-29 Debug IP Packet Field Descriptions

Field	Description
IP:	Indicates that this is an IP packet.
s = 131.108.13.44 (Fddi0)	Indicates the source address of the packet and the name of the interface that received the packet.
d = 157.125.254.1 (Serial2)	Indicates the destination address of the packet and the name of the interface (in this case, S2) through which the packet is being sent out on the network.
g = 131.108.16.2	Indicates the address of the next hop gateway.
forward	Indicates that the router is forwarding the packet. If a filter denies a packet, "access denied" replaces "forward," as shown in the last line of output in Figure 2-54.

The calculation on whether to send a security error message can be somewhat confusing. It depends upon both the security label in the datagram and the label of the incoming interface. First, the label contained in the datagram is examined for anything obviously wrong. If nothing is wrong, assume it to be correct. If there is something wrong, the datagram is treated as *unclassified genser.* Then the label is compared with the interface range, and the appropriate action is taken as Table 2-30 describes.

Table 2-30 Security Actions

Classification	Authorities	Action Taken
Too low	Too low	No Response
	Good	No Response
	Too high	No Response

Classification	Authorities	Action Taken
In range	Too low	No Response
	Good	Accept
	Too high	Send Error
Too high	Too low	No Response
	In range	Send Error
	Too high	Send Error

The security code can only generate a few types of ICMP error messages. The only possible error messages and their meanings follow:

- "ICMP Parameter problem, code 0"—Error at pointer
- "ICMP Parameter problem, code 1"—Missing option
- "ICMP Parameter problem, code 2"—See Note that follows
- "ICMP Unreachable, code 10"—Administratively prohibited

Note The message "ICMP Parameter problem, code 2" identifies a specific error that occurs in the processing of a datagram. This message indicates that the router received a datagram containing a maximum length IP header but no security option. After being processed and routed to another interface, it is discovered that the outgoing interface is marked with "add a security label." Since the IP header is already full, the system cannot add a label and must drop the datagram and return an error message.

When an IP packet is rejected due to an IP security failure, an audit message is sent via DNSIX NAT. Also, any **debug ip packet** output is appended to include a description of the reason for rejection. These reasons can be any of the following:

- No basic
- No basic, no response
- Reserved class
- Reserved class, no response
- Class too low, no response
- Class too high
- Class too high, bad authorities, no response
- Unrecognized class
- Unrecognized class, no response
- Multiple basic
- Multiple basic, no response
- Authority too low, no response
- Authority too high
- Compartment bits not dominated by maximum sensitivity level

- Compartment bits don't dominate minimum sensitivity level
- Security failure: extended security disallowed
- NLESO source appeared twice
- ESO source not found
- Postroute, failed xfc out
- No room to add IPSO

debug ip pim

Use the **debug ip pim** EXEC command to display Protocol Independent Multicast (PIM) packets received and transmitted as well as PIM related events.The **no** form of this command disables debugging output.

debug ip pim [*group*]
no debug ip pim [*group*]

Syntax Description

group	(Optional) Group name or address to monitor a single group's packet activity

Command Mode

EXEC

Usage Guidelines

PIM uses IGMP packets to communicate between routers and advertise reachability information.

Use this command with **debug ip igmp** and **debug ip mrouting** to observe additional multicast routing information.

Sample Display

Figure 2-55 shows sample **debug ip pim** output.

Figure 2-55 Sample Debug IP PIM Output

```
router# debug ip pim 224.2.0.1

PIM: Received Join/Prune on Ethernet1 from 198.92.37.33
PIM: Received Join/Prune on Ethernet1 from 198.92.37.33
PIM: Received Join/Prune on Tunnel0 from 10.3.84.1
PIM: Received Join/Prune on Ethernet1 from 198.92.37.33
PIM: Received Join/Prune on Ethernet1 from 198.92.37.33
PIM: Received RP-Reachable on Ethernet1 from 131.108.20.31
PIM: Update RP expiration timer for 224.2.0.1
PIM: Forward RP-reachability packet for 224.2.0.1 on Tunnel0
PIM: Received Join/Prune on Ethernet1 from 198.92.37.33
PIM: Prune-list (163.221.196.51/32, 224.2.0.1)
PIM: Set join delay timer to 2 seconds for (163.221.0.0/16, 224.2.0.1) on Ethernet1
PIM: Received Join/Prune on Ethernet1 from 198.92.37.6
PIM: Received Join/Prune on Ethernet1 from 198.92.37.33
PIM: Received Join/Prune on Tunnel0 from 10.3.84.1
PIM: Join-list: (*, 224.2.0.1) RP 131.108.20.31
PIM: Add Tunnel0 to (*, 224.2.0.1), Forward state
PIM: Join-list: (13.0.0.0/8, 224.2.0.1)
PIM: Add Tunnel0 to (13.0.0.0/8, 224.2.0.1), Forward state
PIM: Join-list: (128.3.0.0/16, 224.2.0.1)
PIM: Prune-list (198.92.84.16/28, 224.2.0.1) RP-bit set RP 198.92.84.16
PIM: Send Prune on Ethernet1 to 198.92.37.6 for (198.92.84.16/28, 224.2.0.1), RP
PIM: For RP, Prune-list: 128.9.0.0/16
PIM: For RP, Prune-list: 128.16.0.0/16
PIM: For RP, Prune-list: 128.49.0.0/16
```

```
PIM: For RP, Prune-list: 128.84.0.0/16
PIM: For RP, Prune-list: 128.146.0.0/16
PIM: For 10.3.84.1, Join-list: 198.92.84.16/28
PIM: Send periodic Join/Prune to RP via 198.92.37.6 (Ethernet1)
```

Explanations for individual lines of output from Figure 2-55 follow.

The following lines appear periodically when PIM is running in sparse mode and indicate to this router which multicast groups and multicast sources other routers are interested in:

```
PIM: Received Join/Prune on Ethernet1 from 198.92.37.33
PIM: Received Join/Prune on Ethernet1 from 198.92.37.33
```

The following lines appear when a rendezvous point (RP) message is received and the RP timer is reset. The expiration timer sets a checkpoint to make sure the RP still exists; otherwise a new RP must be discovered:

```
PIM: Received RP-Reachable on Ethernet1 from 131.108.20.31
PIM: Update RP expiration timer for 224.2.0.1
PIM: Forward RP-reachability packet for 224.2.0.1 on Tunnel0
```

The prune-list message in the following line states that this router is not interested in the source address information. The prune message tells an upstream router to stop forwarding multicast packets from this source.

```
PIM: Prune-list (163.221.196.51/32, 224.2.0.1)
```

In the following line, a second router on the network wants to override the prune message that the upstream router just received. The timer is set at a random value so that if there are additional routers on the network that still want to receive multicast packets for the group, only one will actually send the message. The other routers will receive the join message and then suppress sending their own message.

```
PIM: Set join delay timer to 2 seconds for (163.221.0.0/16, 224.2.0.1) on Ethernet1
```

In the following line, a join message is sent towards the RP for all sources:

```
PIM: Join-list: (*, 224.2.0.1) RP 131.108.20.31
```

In the following lines, the interface is being added to the outgoing interface (OIF) of the *,G and S,G mroute table entry so that packets from the source will be forwarded out that particular interface:

```
PIM: Add Tunnel0 to (*, 224.2.0.1), Forward state
PIM: Add Tunnel0 to (13.0.0.0/8, 224.2.0.1), Forward state
```

The following line appears in sparse mode only. There are two trees on which data may be received: the RP tree and the source tree. In dense mode there is no RP. After the source and the receiver have discovered one another at the RP, the first-hop router for the receiver will usually join to the source tree rather than the RP tree:

```
PIM: Prune-list (198.92.84.16/28, 224.2.0.1) RP-bit set RP 198.92.84.16
```

The Send Prune message in the next line shows that a router is sending a message to a second router saying that the first router no longer wants to receive multicast packets for the S,G. The “RP” at the end of the message indicates that the router is pruning the RP tree and is most likely joining the source tree, although the router may not have downstream members for the group or downstream routers with members of the group. The output shows which specific sources this router no longer wants to receive multicast from.

```
PIM: Send Prune on Ethernet1 to 198.92.37.6 for (198.92.84.16/28, 224.2.0.1), RP
```

The following lines indicate a prune message is sent toward the RP so that router can join the source tree rather than the RP tree:

```
PIM: For RP, Prune-list: 128.9.0.0/16
PIM: For RP, Prune-list: 128.16.0.0/16
PIM: For RP, Prune-list: 128.49.0.0/16
```

In the following line, a periodic message is sent towards the RP. The default period is once per minute. Prune and join messages are sent toward the RP or source rather than directly to the RP or source. It is the responsibility of the next-hop router to take proper action with this message, such as continuing to forward it to the next router in the tree.

```
PIM: Send periodic Join/Prune to RP via 198.92.37.6 (Ethernet1)
```

Related Commands

debug ip mrouting
debug ip igmp

debug ip rip

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

debug ip rip
no debug ip rip

Syntax Description

This command has no arguments or keywords.

Command Mode

EXEC

Sample Display

Figure 2-56 shows sample **debug ip rip** output.

Figure 2-56 Sample Debug IP RIP Output

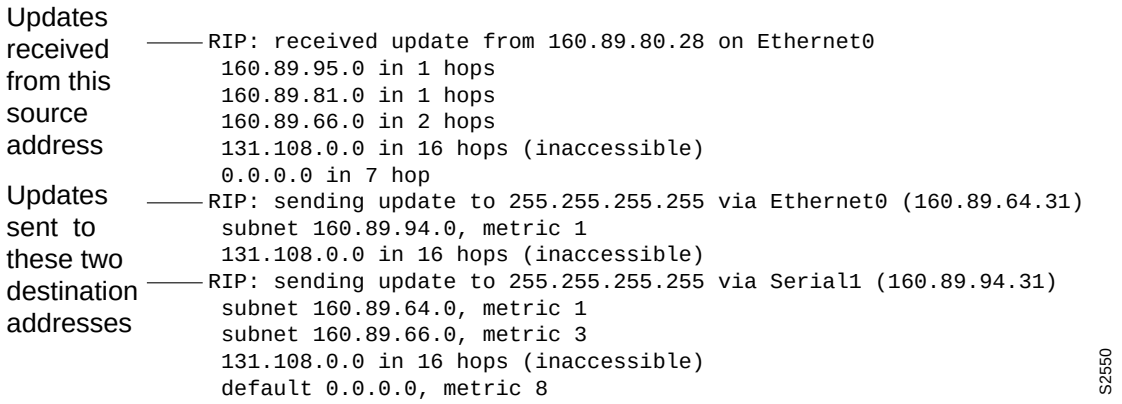


Figure 2-56 shows that the router being debugged has received updates from one router at source address 160.89.80.28. That router sent information about five destinations in the routing table update. Notice that the fourth destination address in the update—131.108.0.0—is inaccessible because it is more than 15 hops away from the router sending the update. The router being debugged also sent updates, in both cases to broadcast address 255.255.255.255 as the destination.

The first line in Figure 2-56 is self-explanatory.

The second line in Figure 2-56 is an example of a routing table update. It shows how many hops a given Internet address is from the router.

The entries in Figure 2-56 show that the router is sending updates that are similar, except that the number in parentheses is the source address encapsulated into the IP header.

Examples of additional output that the **debug ip rip** command can generate follow.

Entries such as the following appear at startup or when an event occurs such as an interface transitioning or a user manually clearing the routing table:

```
RIP: broadcasting general request on Ethernet0
RIP: broadcasting general request on Ethernet1
```

The following line is self-explanatory:

```
RIP: received request from 160.89.80.207 on Ethernet0
```

An entry such as the following is most likely caused by a malformed packet from the transmitter:

```
RIP: bad version 128 from 160.89.80.43
```

debug ip routing

Use the **debug ip routing** EXEC command to display information on Routing Information Protocol (RIP) routing table updates and route-cache updates. The **no** form of this command disables debugging output.

debug ip routing
no debug ip routing

Syntax Description

This command has no arguments or keywords.

Command Mode

EXEC

Sample Display

Figure 2-57 shows sample **debug ip routing** output.

Figure 2-57 Sample Debug IP Routing Output

```
router# debug ip routing

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

Explanations for representative lines of output in Figure 2-57 follow.

In the following lines, a newly created entry has been added to the IP routing table. The “metric change” indicates that this entry existed previously, but its metric changed and the change was reported by means of IGRP. The metric could also be reported via RIP, OSPF, or another IP routing protocol. The numbers inside the brackets report the administrative distance and the actual metric.

"Cache invalidation" means that the fast switching cache was invalidated due to a routing table change. "New version" is the version number of the routing table. When the routing table changes, this number is incremented. The hexadecimal numbers are internal numbers that vary from version to version and software load to software load.

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

In the following output, the "holddown" and "cache invalidation" lines are displayed. Most of the distance vector routing protocols use "holddown" to avoid typical problems like counting to infinity and routing loops. If you look at the output of **show ip protocols** you will see what the timer values are for "holddown" and "cache invalidation". "Cache invalidation" corresponds to "came out of holddown". "Delete route" is triggered when a better path comes along. It gets rid of the old inferior path.

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

debug ip security

Use the **debug ip security** EXEC command to display IP security option processing. The **no** form of this command disables debugging output.

debug ip security
no debug ip security

Syntax Description

This command has no arguments or keywords.

Command Mode

EXEC

Usage Guidelines

The **debug ip security** command displays information for both basic and extended IP security options. For interfaces where **ip security** is configured, each IP packet processed for that interface results in debugging output regardless of whether the packet contains IP security options. IP packets processed for other interfaces that also contain IP security information also trigger debugging output. Some additional IP security debugging information is also controlled by the **debug ip packet** EXEC command.

Note Because the **debug ip security** command generates a significant amount of output for every IP packet processed, use it only when traffic on the IP network is low, so other activity on the system is not adversely affected.

Sample Display

Figure 2-58 shows sample **debug ip security** output.

Figure 2-58 Sample Debug IP Security Output

```
router# debug ip security

IP Security: src 198.92.72.52 dst 198.92.72.53, number of BSO 1
     idb: NULL
     pak: insert (0xFF) 0x0
IP Security: BSO postroute: SECINSERT changed to secret (0x5A) 0x10
IP Security: src 198.92.72.53 dst 198.92.72.52, number of BSO 1
     idb: secret (0x6) 0x10 to secret (0x6) 0x10, no implicit
          def secret (0x6) 0x10
     pak: secret (0x5A) 0x10
IP Security: checking BSO 0x10 against [0x10 0x10]
IP Security: classified BSO as secret (0x5A) 0x10
```

Table 2-31 describes significant fields shown in Figure 2-58.

Table 2-31 Debug IP Security Field Descriptions

Field	Description
number of BSO	Indicates the number of basic security options found in the packet.
idb	Provides information on the security configuration for the incoming interface.
pak	Provides information on the security classification of the incoming packet.
src	Indicates the source IP address.
dst	Indicates the destination IP address.

Explanations for representative lines of output in Figure 2-58 follow.

The following line indicates that the packet was locally generated, and it has been classified with the internally significant security level “insert” (0xff) and authority 0x0:

```
idb: NULL
pak: insert (0xff) 0x0
```

The following line indicates that the packet was received via an interface with dedicated IP security configured. Specifically, the interface is configured at security level “secret” and with authority information of 0x0. The packet itself was classified at level “secret” (0x5a) and authority 0x10.

```
idb: secret (0x6) 0x10 to secret (0x6) 0x10, no implicit
     def secret (0x6) 0x10
pak: secret (0x5A) 0x10
```

debug ip tcp driver

Use the **debug ip tcp driver** EXEC command to display information on Transmission Control Protocol (TCP) driver events; for example, connections opening or closing, or packets being dropped because of full queues. The **no** form of this command disables debugging output.

debug ip tcp driver
no debug ip tcp driver

Syntax Description

This command has no arguments or keywords.

Command Mode

EXEC

Usage Guidelines

The TCP driver is the process that the router software uses to send packet data over a TCP connection. Remote source-route bridging, STUN (serial tunneling), and X.25 switching currently use the TCP driver.

Using the **debug ip tcp driver** command together with the **debug ip tcp driver-pak** command provides the most verbose debugging output concerning TCP driver activity.

Sample Display

Figure 2-59 shows sample **debug ip tcp driver** output.

Figure 2-59 Sample Debug IP TCP Driver Output

```
router# debug ip tcp driver

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

Explanations for individual lines of output from Figure 2-59 follow.

Table 2-32 describes the fields in the first line of output.

Table 2-32 Debug IP TCP Driver Field Descriptions

Field	Description
TCPDRV359CD8:	Unique identifier for this instance of TCP driver activity.
Active open 160.89.80.26	Indication that the router at IP address 160.89.80.26 has initiated a connection to another router.
:0	The TCP port number the initiator of the connection uses to indicate that any port number can be used to set up a connection.
--> 160.89.80.25	The IP address of the remote router to which the connection has been initiated.

Field	Description
:1996	The TCP port number that the initiator of the connection is requesting that the remote router use for the connection. (1996 is a private TCP port number reserved in this implementation for remote source-route bridging.)
OK,	Indication that the connection has been established. If the connection has not been established, this field and the following field do not appear in this line of output.
lport 36628	The TCP port number that has actually been assigned for the initiator to use for this connection.

The following line indicates that the TCP driver user (remote source-route bridging, in this case) will allow TCP to drop the connection if excessive retransmissions occur:

```
TCPDRV359CD8: enable tcp timeouts
```

The following line indicates that the TCP driver user (in this case, remote source-route bridging) at IP address 160.89.80.26 (and using TCP port number 36628) is requesting that the connection to IP address 160.89.80.25 using TCP port number 1996 be aborted:

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

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

```
TCPDRV359CD8: 160.89.80.26:36628 --> 160.89.80.25:1996 DoClose tcp abort
```

debug ip tcp driver-pak

Use the **debug ip tcp driver-pak** EXEC command to display information on every operation that the Transmission Control Protocol (TCP) driver performs. The **no** form of this command disables debugging output.

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

Syntax Description

This command has no arguments or keywords.

Command Mode

EXEC

Usage Guidelines

This command turns on a verbose debugging by logging at least one debugging message for every packet sent or received on the TCP driver connection.

The TCP driver is the process that the router software uses to send packet data over a TCP connection. Remote source-route bridging, STUN (serial tunneling), and X.25 switching currently use the TCP driver.

To observe the context within which certain **debug ip tcp driver-pak** messages occur, turn on this command in conjunction with the **debug ip tcp driver** command.

Note Because the **debug ip tcp driver-pak** command generates so many messages, use it only on lightly loaded systems. This command not only places a significant load on the system processor, but it may even change the symptoms of any unexpected behavior that occur.

Sample Display

Figure 2-60 shows sample **debug ip tcp driver-pak** output.

Figure 2-60 Sample Debug IP TCP Driver-Pak Output

```
router# debug ip tcp driver-pak

TCPDRV359CD8: send 2E8CD8 (len 26) queued
TCPDRV359CD8: output pak 2E8CD8 (len 26) (26)
TCPDRV359CD8: readf 42 bytes (Thresh 16)
TCPDRV359CD8: readf 26 bytes (Thresh 16)
TCPDRV359CD8: readf 10 bytes (Thresh 10)
TCPDRV359CD8: send 327E40 (len 4502) queued
TCPDRV359CD8: output pak 327E40 (len 4502) (4502)
```

Explanations for individual lines of output from Figure 2-60 follow.

Table 2-33 describes the fields shown in the first line of output.

Table 2-33 Debug TCP Driver-Pak Field Descriptions

Field	Description
TCPDRV359CD8	Unique identifier for this instance of TCP driver activity.
send	Indication that this event involves the TCP driver sending data.
2E8CD8	Address in memory of the data the TCP driver is sending.
(len 26)	Length of the data (in bytes).
queued	Indication that the TCP driver user process (in this case, remote source-route bridging) has transferred the data to the TCP driver to send.

The following line indicates that the TCP driver has sent the data that it had received from the TCP driver user, as shown in the previous line of output. The last field in the line (26) indicates that the 26 bytes of data were sent out as a single unit.

```
TCPDRV359CD8: output pak 2E8CD8 (len 26) (26)
```

The following line indicates that the TCP driver has received 42 bytes of data from the remote IP address. The TCP driver user (in this case, remote source-route bridging) has established an input threshold of 16 bytes for this connection. (The input threshold instructs the TCP driver to transfer data to the TCP driver user only when at least 16 bytes are present.)

```
TCPDRV359CD8: readf 42 bytes (Thresh 16)
```

debug ip tcp transactions

Use the **debug ip tcp transactions** EXEC command to display information on significant Transmission Control Protocol (TCP) transactions such as state changes, retransmissions, and duplicate packets. The **no** form of this command disables debugging output.

debug ip tcp transactions
no debug ip tcp transactions

Syntax Description

This command has no arguments or keywords.

Command Mode

EXEC

Usage Guidelines

This command is particularly useful for debugging a performance problem on a TCP/IP network that you have isolated above the data link layer.

The **debug ip tcp transactions** command displays output for packets the router sends and receives, but does not display output for packets it forwards.

Sample Display

Figure 2-61 shows sample **debug ip tcp transactions** output.

Figure 2-61 Sample Debug IP TCP Output

```
router# debug ip tcp transactions

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

Table 2-34 describes significant fields shown in Figure 2-61.

Table 2-34 Debug IP TCP Field Descriptions

Field	Description
TCP:	Indicates that this is a TCP transaction.
sending SYN	Indicates that a synchronize packet is being sent.
seq 168108	Indicates the sequence number of the data being sent.

Field	Description
ack 88655553	Indicates the sequence number of the data being acknowledged.
TCP0:	Indicates the TTY number (0, in this case) with which this TCP connection is associated.
Connection to 26.9.0.13:22530	Indicates the remote address with which a connection has been established.
advertising MSS 966	Indicates the maximum segment size this side of the TCP connection is offering to the other side.
state was LISTEN -> SYNSENT	Indicates that the TCP state machine changed state from LISTEN to SYNSENT. Possible TCP states follow: CLOSED—Connection closed. CLOSEWAIT—Received a FIN segment. CLOSING—Received a FIN/ACK segment. ESTAB—Connection established. FINWAIT 1—Sent a FIN segment to start closing the connection. FINWAIT 2—Waiting for a FIN segment. LASTACK—Sent a FIN segment in response to a received FIN segment. LISTEN—Listening for a connection request. SYNRCVD—Received a SYN segment, and responded. SYNSENT—Sent a SYN segment to start connection negotiation. TIMEWAIT—Waiting for network to clear segments for this connection before the network no longer recognizes the connection as valid. This must occur before a new connection can be set up.
[23 -> 26.9.0.13(22530)]	Within these brackets: The first field (23) indicates local TCP port. The second field (26.9.0.13) indicates the destination IP address. The third field (22530) indicates the destination TCP port.
restart retransmission in 5996	Indicates the number of milliseconds until the next retransmission takes place.

debug ipx ipxwan

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

debug ipx ipxwan
no debug ipx ipxwan

Syntax Description

This command has no arguments or keywords.

Command Mode

EXEC

Usage Guidelines

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

Sample Display

Figure 2-62 shows sample **debug ipx ipxwan** output during link startup.

Figure 2-62 Sample Debug IPX IPXWAN Output

```
router# debug ipx ipxwan

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

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

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

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

The following line indicates that the interface has initialized:

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

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

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

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

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

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

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

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

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

debug ipx packet

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

debug ipx packet
no debug ipx packet

Syntax Description

This command has no arguments or keywords.

Command Mode

EXEC

Usage Guidelines

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

Note In order to generate **debug ipx packet** information on all IPX traffic traveling over the router, you must first configure the router so that fast switching is disabled. Use the **no ipx route-cache** command on all interfaces on which you want to observe traffic. If the router is configured for IPX fast switching, only non-fast switched packets will produce output. When the IPX cache is invalidated or cleared, one packet for each destination is displayed as the cache is repopulated.

Sample Display

Figure 2-63 shows sample **debug ipx packet** output.

Figure 2-63 Sample Debug IPX Packet Output

```
router# debug ipx packet

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

In Figure 2-63, the first line indicates that the router receives a packet from a Novell station (address 160.0260.8c4c.4f22); this trace does not indicate the address of the immediate router sending the packet to this router. In the second line, the router forwards the packet toward the Novell server (address 1.0000.0000.0001) through an immediate router (183.0000.0c01.5d85).

Table 2-35 describes significant fields shown in Figure 2-63.

Table 2-35 Debug IPX Packet Field Descriptions

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

debug ipx routing

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

debug ipx routing
no debug ipx routing

Syntax Description

This command has no arguments or keywords.

Command Mode

EXEC

Usage Guidelines

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

Sample Display

Figure 2-64 shows sample **debug ipx routing** output.

Figure 2-64 Sample Debug IPX Routing Output

```
router# debug ipx routing

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

Table 2-36 describes significant fields shown in Figure 2-64.

Table 2-36 Debug IPX Routing Field Descriptions

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

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

Related Command

debug ipx sap

debug ipx sap

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

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

Syntax Description

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

Command Mode

EXEC

Usage Guidelines

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

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

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

Sample Display

Figure 2-65 shows sample **debug ipx sap** output.

Figure 2-65 Sample Debug IPX SAP Output

```
router# debug ipx sap

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

Describes a single SAP packet

S2551

As Figure 2-65 shows, the **debug ipx sap** command generates multiple lines of output for each SAP packet—a packet summary message and a service detail message.

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

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

Table 2-37 describes the fields shown in the second line of output in Figure 2-65.

Table 2-37 Debug IPX SAP Field Descriptions—Part 1

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

Table 2-38 describes the fields shown in the third and fourth lines of output in Figure 2-65.

Table 2-38 Debug IPX SAP Field Descriptions—Part 2

Field	Description
type 0x4	Indicates the type of service the server sending the packet provides. Format is 0x*n*. Some of the values for *n* are proprietary to Novell. Those values for *n* that have been published include 0—Unknown 1—User 2—User group 3—Print queue 4—File server 5—Job server 6—Gateway 7—Print server 8—Archive queue 9—Archive server A—Job queue B—Administration 21—NAS SNA gateway 24—Remote bridge server 2D—Time Synchronization VAP 2E—Dynamic SAP 47—Advertising print server 4B—Btrieve VAP 5.0 4C—SQL VAP 7A—TES—NetWare for VMS 98—NetWare access server 9A—Named Pipes server 9E—Portable NetWare—UNIX 111—Test server 166—NetWare management 233—NetWare management agent 237—NetExplorer NLM 239—HMI hub 23A—NetWare LANalyzer agent 26A—NMS management FFFF—Wildcard (any SAP service) Contact Novell for more information.
“HELLO2”	Name of the server being advertised.
199.0002.0004.0006 (451)	Indicates the network number and address (and socket) of the server generating the SAP packet.
2 hops	Number of hops to the server from the router.

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

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

As Figure 2-65 shows, the format for **debug ipx sap** output describing a SAP update the router sends is similar to that describing a SAP update the router receives, except that the ssoc: field replaces the src: field, as the following line of output indicates:

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

Table 2-39 describes possible values for the ssoc: field.

Table 2-39 Debug IPX SAP Field Descriptions—Part 3

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

Related Command

debug ipx routing

debug isdn-event

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

debug isdn-event
no debug isdn-event

Syntax Description

This command has no arguments or keywords.

Command Mode

EXEC

Usage Guidelines

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

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

Sample Display

Figure 2-66 shows sample **debug isdn-event** output of call setup events for an outgoing call.

Figure 2-66 Sample Debug ISDN-Event Output—Call Setup Outgoing Call

```
router# debug isdn-event

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

Figure 2-67 shows sample **debug isdn-event** output of call setup events for an incoming call. The values used for internal purposes are unpacked information elements. The values that follow the ISDN specification are an interpretation of the unpacked information elements. Refer to the "ISDN Switch Types, Codes, and Values" appendix for information about these values.

Figure 2-67 Sample Debug ISDN-Event Output—Call Setup Incoming Call

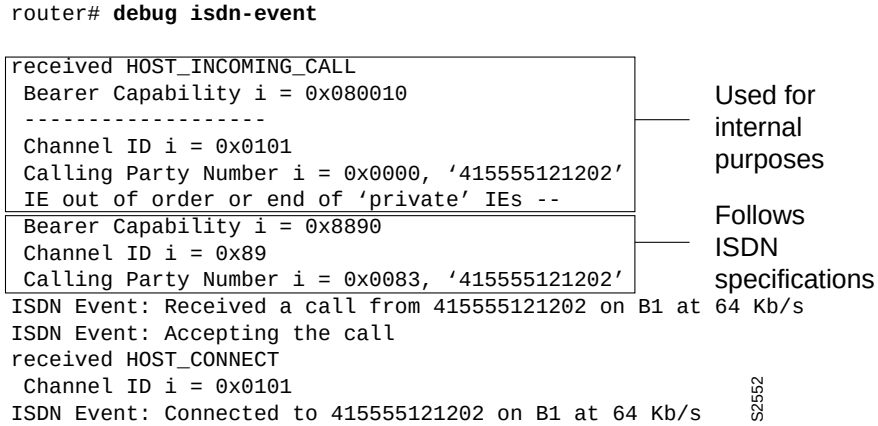


Figure 2-68 shows sample **debug isdn-event** output of call teardown events for a call that has been hung up by the other side of the connection.

Figure 2-68 Sample Debug ISDN-Event Output—Call Teardown by Far End

```
router# debug isdn-event

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

Figure 2-69 shows sample **debug isdn-event** output of a call teardown event for an outgoing or incoming call that has been hung up by the ISDN interface on the router side.

Figure 2-69 Sample Debug ISDN-Event Output—Call Teardown Local Side

```
router# debug isdn-event

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

Table 2-40 describes significant fields shown in Figure 2-66 through Figure 2-69.

Table 2-40 Debug ISDN-Event Field Descriptions

Field	Description
Bearer Capability	Indicates the requested bearer service to be provided by the network.
i=	Indicates the Information Element Identifier. The value depends on the field it is associated with. Refer to the ITU-T[1] Q.931 specification for details about the possible values associated with each field for which this identifier is relevant.
Channel ID	Indicates the Channel Identifier. The value 83 indicates any channel, 0101 indicates the B1 channel, and 89 indicates the B1 channel.
Calling Party Number	Identifies the called party. This field is only present in outgoing calls. Note that it may be replaced by the Keypad facility field. This field uses the IA5 character set.
IE out of order or end of 'private' IEs	Indicates that an information element identifier is out of order or there are no more private network information element identifiers to interpret.
Received a call from 415555121202 on B1 at 64Kb/s	Identifies the origin of the call. This field is present only in incoming calls. Note that the information about the incoming call includes the channel and speed. Whether this number is displayed depends on the network delivering the calling party number.

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

Figure 2-70 shows sample **debug isdn-event** output of a call teardown event for a call that has passed call screening then has been hung up by the ISDN interface on the far end side.

Figure 2-70 Sample Debug ISDN-Event—Call Screening Normal Disconnect

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

Figure 2-71 shows sample **debug isdn-event** output of a call teardown event for a call that has not passed call screening and has been rejected by the ISDN interface on the router side.

Figure 2-71 Sample Debug ISDN-Event—Call Screening Call Rejection

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

Figure 2-72 shows sample **debug isdn-event** output of a call teardown event for an outgoing call that uses a dialer subaddress.

Figure 2-72 Sample Debug ISDN-Event Display—Called Party Subaddress

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

debug isdn-q921

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

debug isdn-q921
no debug isdn-q921

Syntax Description

This command has no arguments or keywords.

Command Mode

EXEC

Usage Guidelines

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

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

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

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

Sample Display

Figure 2-73 shows sample **debug isdn-q921** output for an outgoing call.

Figure 2-73 Sample Debug ISDN-Q921 Output for Outgoing Call

```
router# debug isdn-q921

471.348 TX -> RRp sapi = 0 tei = 67 nr = 19
471.372 RX <- RRp sapi = 0 tei = 67 nr = 17
471.376 TX -> RRf sapi = 0 tei = 67 nr =19
471.388 RX <- RRf sapi = 0 tei = 67 nr = 17
471.968 TX -> INFOc sapi = 0 tei = 67 ns = 17 nr = 19 i = 0x08010505040028890180183
700A80353535313231323032
472.068 RX <- RRr sapi = 0 tei = 67 nr = 18
472.088 RX <- INFOc sapi = 0 tei = 67 ns = 19 nr = 18 i = 0x0801850202180189
472.096 TX -> RRr sapi = 0 tei = 67 nr = 20
472.268 RX <- INFOc sapi = 0 tei = 67 ns = 20 nr 18 i = 0x08018507
472.276 TX -> RRr sapi = 0 tei = 67 nr = 21
472.284 TX -> INFOc sapi = 0 tei = 67 ns 18 nr = 21 i = 0x0801050F
472.356 RX <- RRr sapi = 0 tei = 67 nr = 19
```

Call Setup message

Call Proceeding message

Call Connect message

Connect Ack message

S2555

Figure 2-74 shows sample **debug isdn-q921** output for a startup message on a DMS-100 switch.

Figure 2-74 Sample Debug ISDN-Q921 Output for Startup Message on a DMS-100 Switch

```
router# debug isdn-q921

139.516 TX -> IDREQ ri = 48386 ai = 127
139.520 RX <- IDREM ri = 0 ai = 89
139.544 RX <- IDASSN ri = 48386 ai = 90
139.552 TX -> SABMEp sapi = 0 tei = 90
139.552 RX <- IDCKRQ ri = 0 ai = 127
139.560 TX -> IDCKRP ri = 36131 ai = 90
140.548 RX <- IDCKRQ ri = 0 ai = 127
140.556 TX -> IDCKRP ri = 24404 ai = 90
140.560 TX -> SABMEp sapi = 0 tei = 90
140.584 RX <- UAf sapi = 0 tei = 90
140.592 TX -> INFOc sapi = 0 tei = 90 ns = 0 nr = 0
 INFORMATION pd = 8 callref = (null)
SPID Information i = 0x343135393033383336363031
140.624 RX <- RRr sapi = 0 tei = 90 nr = 1
140.592 RX <- INFOc sapi = 0 tei = 90 ns = 0 nr = 0
 INFORMATION pd = 8 callref = (null)
ENDPOINT IDent i = 0xF080
140.768 TX -> RRr sapi = 0 tei = 90 nr = 1
150.768 TX -> RRp sapi = 0 tei = 90 nr = 1
150.788 RX <- RRf sapi = 0 tei = 90 nr = 1
160.796 TX -> RRp sapi = 0 tei = 90 nr = 1
160.816 RX <- RRf sapi = 0 tei = 90 nr = 1
```

L2 link establishment

S2556

Figure 2-75 shows sample **debug isdn-q921** output for an incoming call. It is an incoming SETUP message that assumes the L2 link is already established to the other side.

Figure 2-75 Debug ISDN-Q921 Output for Incoming Call

```
router# debug isdn-q921

234423.764 TX -> RRp sapi = 0 tei = 66 nr = 36
234423.780 RX <- RRp sapi = 0 tei = 66 nr = 26
234423.784 TX -> RRf sapi = 0 tei = 66 nr = 36
234423.808 RX <- RRf sapi = 0 tei = 66 nr = 26
234425.800 RX <- UAf sapi = 0 tei = 127 i =
0x08010805040288900180018001896C10008338313031323334453637383930 32
234425.820 TX -> INFOc sapi = 0 tei = 66 ns = 36 nr = 36 i=0x08018807
234425.904 RX <- RRr sapi = 0 tei = 90 nr = 27
234425.920 RX <- INFOc sapi = 0 tei = 66 ns = 36 nr = 33 i=0x0801080F
234433.936 TX -> RRr sapi = 0 tei = 66 nr = 37
234435.940 RX <- RRp sapi = 0 tei = 66 nr = 27
234435.980 TX -> RRf sapi = 0 tei = 66 nr = 37
234435.640 RX <- RRf sapi = 0 tei = 66 nr = 27
```

Table 2-41 describes significant fields in Figure 2-73, Figure 2-74, and Figure 2-75.

Table 2-41 Debug ISDN-Q921 Field Descriptions

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

debug isdn-q931

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

debug isdn-q931
no debug isdn-q931

Syntax Description

This command has no arguments or keywords.

Command Mode

EXEC

Usage Guidelines

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

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

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

Sample Display

Figure 2-76 shows sample **debug isdn-q931** output of a call setup procedure for an outgoing call.

Figure 2-76 Sample Debug ISDN-Q931 Output—Call Setup Procedure for an Outgoing Call

```
router# debug isdn-q931

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

Figure 2-77 shows sample **debug isdn-q931** output of a call setup procedure for an incoming call.

Figure 2-77 Sample Debug ISDN-Q931 Output—Call Setup Procedure for an Incoming Call

```
router# debug isdn-q931

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

Figure 2-78 shows sample **debug isdn-q931** output of a call teardown procedure from the network.

Figure 2-78 Sample Debug ISDN-Q931 Output—Call Teardown Procedure from the Network

```
router# debug isdn-q931

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

Figure 2-79 shows sample **debug isdn-q931** output of a call teardown procedure from the router.

Figure 2-79 Sample Debug ISDN-Q931 Output—Call Teardown Procedure from the Router

```
router# debug isdn-q931

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

Table 2-42 describes significant fields in Figure 2-76 through Figure 2-79.

Table 2-42 Debug ISDN-Q931 Call Setup Procedure Field Descriptions

Field	Description
234191.372	Indicates the time, in seconds, at which the message was transmitted from or received by the network layer on the router. The time is maintained by an internal clock. This internal clock is used for timeout purposes and timestamping.
TX	Indicates that this message is being transmitted from the local router (user side) to the network side of the ISDN interface.
RX	Indicates that this message is being received by the user side of the ISDN interface from the network side.

Field	Description
SETUP	Indicates that the SETUP message type has been sent to initiate call establishment between peer network layers. This message can be sent from either the local router or the network.
pd	Indicates the protocol discriminator. The protocol discriminator distinguishes messages for call control over the user-network ISDN interface from other ITU-T-defined messages, including other Q.931messages. The protocol discriminator is 8 for call control messages such as SETUP. For basic-1tr6, the protocol discriminator is 65.
callref	Indicates the call reference number in hexadecimal. The value of this field indicates the number of calls made from either the router (outgoing calls) or the network (incoming calls). Note that the originator of the SETUP message sets the high-order bit of the call reference number to 0. The destination of the connection sets the high-order bit to 1 in subsequent call control messages, such as the CONNECT message. For example, callref = 0x04 in the request becomes callref = 0x84 in the response.
Bearer Capability	Indicates the requested bearer service to be provided by the network.
i=	Indicates the Information Element Identifier. The value depends on the field it is associated with. Refer to the ITU-T Q.931 specification for details about the possible values associated with each field for which this identifier is relevant.
Channel ID	Indicates the Channel Identifier. The value 83 indicates any channel, 89 indicates the B1 channel, and 8A indicates the B2 channel. For more information about the Channel Identifier, refer to ITU-T Recommendation Q.931.
Called Party Number	Identifies the called party. This field is only present in outgoing SETUP messages. Note that it can be replaced by the Keypad facility field. This field uses the IA5 character set.
Calling Party Number	Identifies the origin of the call. This field is present only in incoming SETUP messages. This field uses the IA5 character set.
CALL_PROC	Indicates the CALL PROCEEDING message; the requested call setup has begun and no more call setup information will be accepted.
CONNECT	Indicates that the called user has accepted the call.
CONNECT_ACK	Indicates that the calling user acknowledges the called user's acceptance of the call.
DISCONNECT	Indicates either that the user side has requested the network to clear an end-to-end connection or that the network has cleared the end-to-end connection.
Cause	Indicates the cause of the disconnect. Refer to ITU-T recommendation Q.931 for detailed information about DISCONNECT cause codes and RELEASE cause codes.
Looking Shift to Codeset 6	Indicates that the next information elements will be interpreted according to information element identifiers assigned in codeset 6. Codeset 6 means that the information elements are specific to the local network.
Codeset 6 IE 0x1 i = 0x82, '10'	Indicates charging information. This information is specific to the NTT switch type and may not be sent by other switch types.

Field	Description
RELEASE	Indicates that the sending equipment will release the channel and call reference. The recipient of this message should prepare to release the call reference and channel.
RELEASE_COMP	Indicates that the sending equipment has received a RELEASE message and has now released the call reference and channel.

debug isis adj packets

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

debug isis adj packets
no debug isis adj packets

Syntax Description

This command has no arguments or keywords.

Command Mode

EXEC

Sample Display

Figure 2-80 shows sample **debug isis adj packets** output.

Figure 2-80 Sample Debug ISIS Adj Packets Output

```
router# debug isis adj packets

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

Explanations for individual lines of output from Figure 2-80 follow.

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

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

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

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

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

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

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

debug isis spf statistics

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

debug isis spf statistics
no debug isis spf statistics

Syntax Description

This command has no arguments or keywords.

Command Mode

EXEC

Usage Guidelines

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

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

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

Sample Display

Figure 2-81 shows sample **debug isis spf statistics** output.

Figure 2-81 Sample Debug ISIS SPF Statistics Output

```
router# debug isis spf packets

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

Table 2-43 describes significant fields shown in Figure 2-81.

Table 2-43 Debug ISDN-Event Field Descriptions

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

Explanations for individual lines of output from Figure 2-81 follow.

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

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

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

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

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

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

debug isis update-packets

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

debug isis update-packets
no debug isis update-packets

Syntax Description

This command has no arguments or keywords.

Command Mode

EXEC

Sample Display

Figure 2-82 shows sample **debug isis update-packets** output.

Figure 2-82 Sample Debug ISIS Update-Packets Output

```
router# debug isis update-packets

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

Explanations for individual lines of output from Figure 2-82 follow.

The following lines indicate that the router has sent a periodic Level 1 and Level 2 complete sequence number PDU on Ethernet 0:

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

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

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

The following lines indicate that the NSAP identified as 8888.8800.0181.00 was added to the Level 2 LSP 1600.8906.4022.00-00. The new sequence number associated with this LSP is 0x10.

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

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

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

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

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

The following lines indicate that a Level 2 partial sequence number PDU (PSNP) has been received on Tunnel0:

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

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

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

debug lapb

Use the **debug lapb** EXEC command to display all traffic for interfaces using Link Access Protocl, Balanced (LAPB) encapsulation. The **no** form of this command disables debugging output.

debug lapb
no debug lapb

Syntax Description

This command has no arguments or keywords.

Command Mode

EXEC

Usage Guidelines

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

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

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

Sample Display

Figure 2-83 shows sample **debug lapb** output. (The numbers 1 through 7 at the top of the display have been added in order to aid documentation.)

Figure 2-83 Sample Debug LAPB Output

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

In Figure 2-83 each line of output describes a LAPB event. There are two types of LAPB events: frame events (when a frame enters or exits the LAPB) and timer events. In Figure 2-83, the last line describes a timer event; all of the other lines describe frame events. Table 2-44 describes the first seven fields shown in Figure 2-83.

Table 2-44 Debug LAPB Field Descriptions

Field	Description
First field	Interface type and unit number reporting the frame event.
Second field	Protocol providing the information.
Third field	Frame event type. Possible values follow: I—Frame input O—Frame output T1—T1 timer expired T3—Interface outage timer expired T4—Idle link timer expired
Fourth field	State of the protocol when the frame event occurred. Possible values follow: BUSY (RNR frame received) CONNECT DISCONNECT DISCSENT (disconnect sent) ERROR (FRMR frame sent) REJSENT (reject frame sent) SABMSENT (SABM frame sent)
Fifth field	In a frame event, this value is the size of the frame (in bytes). In a timer event, this value is the current timer value (in milliseconds).
Sixth field	In a frame event, this value is the frame type name. Possible values for frame type names follow: DISC—Disconnect DM—Disconnect mode FRMR—Frame reject IFRAME—Information frame ILLEGAL—Illegal LAPB frame REJ—Reject RNR—Receiver not ready RR—Receiver ready SABM—Set asynchronous balanced mode SABME—Set asynchronous balanced mode, extended UA—Unnumbered acknowledgment In a T1 timer event, this value is the number of retransmissions already attempted.

Field	Description
Seventh field (Note that this field will not print if the frame control field is required to appear as either a command or a response, and that frame type is correct.)	This field is only present in frame events. It describes the frame type identified by the LAPB address and Poll/Final bit. Possible values are as follows: (C)—Command frame (R)—Response frame P—Command/Poll frame F—Response/Final frame /ERR—Command/Response type is invalid for the control field. An ?ERR generally means that the DTE/DCE assignments are not correct for this link. BAD-ADDR—Address field is neither Command nor Response

A timer event only displays the first six fields of **debug lapb** output. For frame events, however, the fields that follow the sixth field document the LAPB control information present in the frame. Depending on the value of the frame type name shown in the sixth field, these fields may or may not appear. Descriptions of the fields following the first six fields shown in Figure 2-83 follow.

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

For information frames, the value of the N(S) field and the N(R) field will be printed. The N(S) field of an information frame is the sequence number of that frame, so this field will rotate between 0 and 7 for (modulo 8 operation) or 0 and 127 (for modulo 128 operation) for successive outgoing information frames and (under normal circumstances) also will rotate for incoming information frame streams. The N(R) field is a “piggybacked” acknowledgment for the incoming information frame stream; it informs the other end of the link what sequence number is expected next.

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

For the FRMR frame, the error information is decoded to display the rejected control field, V(R) and V(S) values, the Response/Command flag, and the error flags WXYZ.

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

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

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

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

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

```
Serial2: LAPB T1 SABMSENT 1026508 1
Serial2: LAPB O SABMSENT (2) SABM P
```

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

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

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

debug lat packet

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

debug lat packet
no debug lat packet

Syntax Description

This command has no arguments or keywords.

Command Mode

EXEC

Usage Guidelines

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

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

Sample Display

Figure 2-84 shows sample **debug lat packet** output.

Figure 2-84 Sample Debug LAT Packet Output

```
router# debug lat packet

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

The second line of output in Figure 2-84 describes a packet that is input to the router. Table 2-45 describes the fields in this line.

Table 2-45 Debug LAT Packet Field Descriptions

Field	Description
LAT:	Indicates that this display shows LAT debugging output.
I	Indicates that this line of output describes a packet that is input to the router (I) or output from the router (O).
int = Ethernet0	Indicates the interface on which the packet event took place.
src = 0800.2b11.2d13	Indicates the source address of the packet.
dst = 0000.0c01.7876	Indicates the destination address of the packet.

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

The third line of output in Figure 2-84 describes a packet that is output from the router. Table 2-46 describes the last three fields in this line.

Table 2-46 Debug LAT Packet Field Descriptions

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

debug lex rcmd

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

debug lex rcmd
no debug lex rcmd

Syntax Description

This command has no arguments or keywords.

Command Mode

EXEC

Sample Display

Figure 2-85 shows sample **debug lex rcmd** output.

Figure 2-85 Sample Debug LEX Rcmd Output

```
router# debug lex rcmd

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

Explanations for individual lines of output from Figure 2-85 follow.

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

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

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

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

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

```
LEX-RCMD: Lex0 : "inventory" command received
Rcvd rcmd: FF 03 80 41 41 13 00 1A 8A 00 00 16 01 FF 00 00
```

```
Rcvd rcmd: 00 02 00 00 07 5B CD 15 00 00 0C 01 15 26
```

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

debug lnm events

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

debug lnm events
no debug lnm events

Syntax Description

This command has no arguments or keywords.

Command Mode

EXEC

Sample Display

Figure 2-86 shows sample **debug lnm events** output.

Figure 2-86 Sample Debug LNM Events Output

```
router# debug lnm events

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

Explanations for the messages shown in Figure 2-86 follow.

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

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

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

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

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

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

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

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

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

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

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

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

The following message indicates that the router is beginning its attempt to determine whether any stations left the ring during the automatic recovery process for the last beaconing failure. The router attempts to contact stations that were part of the fault domain to detect whether they are still operating on the ring.

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

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

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

The following message indicates that the router is attempting to determine whether any stations left the ring during the automatic recovery process for the last beaconing failure. It is initiating another attempt to contact the two stations in the fault domain.

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

The following message indicates that the router has attempted to determine whether any stations left the ring during the automatic recovery process for the last beaconing failure. It has successfully heard back from both stations that were part of the fault domain.

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

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

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

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

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

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

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

```
IBMNM3: Adapters experiencing congestion
```

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

```
IBMNM3: Beaconing, permanent
```

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

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

In the preceding line of output, the following can replace “Started”: “Next State”, “Finished”, “Timed out”, and “Cannot find station *n*”.

debug lnm llc

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

debug lnm llc
no debug lnm llc

Syntax Description

This command has no arguments or keywords.

Command Mode

EXEC

Usage Guidelines

One line is displayed for each message sent or received.

Sample Display

Figure 2-87 shows sample **debug lnm llc** output.

Figure 2-87 Sample Debug LNM LLC Output

```
router# debug lnm llc

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

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

As Figure 2-87 indicates, **debug lnm llc** output can vary somewhat in format. Table 2-47 describes significant fields shown in the first line of output in Figure 2-87.

Table 2-47 Debug LNM LLC Field Descriptions

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

Explanations for other types of messages shown in Figure 2-87 follow.

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

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

The following message is self-explanatory:

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

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

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

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

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

debug lnm mac

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

debug lnm mac
no debug lnm mac

Syntax Description

This command has no arguments or keywords.

Command Mode

EXEC

Usage Guidelines

One line is displayed for each message sent or received.

Sample Display

Figure 2-88 shows sample **debug lnm mac** output.

Figure 2-88 Sample Debug LNM MAC Output

```
router# debug lnm mac

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

Table 2-48 describes significant fields shown in the first line of output in Figure 2-88.

Table 2-48 Debug LNM MAC Field Descriptions

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

As Figure 2-88 indicates, all **debug lnm mac** messages follow the format described in Table 2-48 except the following:

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

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

debug local-ack state

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

debug local-ack state
no debug local-ack state

Syntax Description

This command has no arguments or keywords.

Command Mode

EXEC

Sample Display

Figure 2-89 shows sample **debug local-ack state** output.

Figure 2-89 Sample Debug Local-Ack State Output

```
router# debug local-ack state

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

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

Table 2-49 Debug Local-Ack State Field Descriptions

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

debug netbios-name-cache

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

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

Syntax Description

This command has no arguments or keywords.

Command Mode

EXEC

Usage Guidelines

Examine the display to diagnose problems in NetBIOS name caching.

Sample Display

Figure 2-90 illustrates a collection of sample **debug netbios-name-cache** output listings.

Figure 2-90 Sample Debug NetBIOS-Name-Cache Output

```
router# debug netbios-name-cache

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

Note The sample display in Figure 2-90 is a composite output. Debugging output that you actually see would not necessarily occur in this sequence.

Table 2-50 describes selected **debug netbios-name-cache** output fields.

Table 2-50 Debug NetBIOS-Name-Cache Field Descriptions

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

The following discussion briefly outlines each line shown in the example provided in Figure 2-90.

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

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

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

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

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

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

The following line indicates that the NetBIOS name ORINDA is in the name cache table and was updated to the current value:

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

debug packet

Use the **debug packet** EXEC command to display information on packets that the network can not classify. The **no** form of this command disables debugging output.

debug packet
no debug packet

Syntax Description

This command has no arguments or keywords.

Command Mode

EXEC

Sample Display

Figure 2-91 shows sample **debug packet** output. Notice how similar it is to **debug broadcast** output.

Figure 2-91 Sample Debug Packet Output

```
router# debug packet

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

Table 2-51 describes significant fields shown in Figure 2-91.

Table 2-51 Debug Packet Field Descriptions

Field	Description
Ethernet0	Name of the Ethernet interface that received the packet.
Unknown	The network could not classify this packet. Examples include packets with unknown link types.
ARPA	This packet uses ARPA-style encapsulation. Possible encapsulation styles vary depending on the media command mode (MCM) and encapsulation style, as follows: **Ethernet (MCM)** *Encapsulation Style* APOLLO ARP ETHERTALK ISO1 ISO3 LLC2 NOVELL-ETHER SNAP

Field	Description
	FDDI (MCM) *Encapsulation Style* APOLLO ISO1 ISO3 LLC2 SNAP
	Frame Relay *Encapsulation Style* BRIDGE FRAME-RELAY
	Serial (MCM) *Encapsulation Style* BFEX25 BRIDGE DDN-X25 DDNX25-DCE ETHERTALK FRAME-RELAY HDLC HDH LAPB LAPBDCE MULTI-LAPB PPP SDLC-PRIMARY SDLC-SECONDARY SLIP SMDS STUN X25 X25-DCE
	Token Ring (MCM) *Encapsulation Style* 3COM-TR ISO1 ISO3 MAC LLC2 NOVELL-TR SNAP VINES-TR
src 0000.0c00.6fa4	MAC address of the node generating the packet.
dst.ffff.ffff.ffff	MAC address of the destination node for the packet.
type 0x0a0	Packet type.
data ...	First 12 bytes of the datagram following the MAC header.
len 60	Length of the message in bytes that the interface received from the wire.
size 64	Length of the message in bytes that the interface received from the wire. Equivalent to the len field.
flags 0x0F00	HDLC or PP flags field.

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

debug ppp

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

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

Syntax Description

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

Command Mode

EXEC

Usage Guidelines

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

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

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

Sample Displays

Figure 2-92 shows sample **debug ppp packet** output as seen from the Link Quality Monitor (LQM) side of the connection. This display example depicts packet exchanges under normal PPP operation.

Figure 2-92 Sample Debug PPP Packet Output

```
router# debug ppp packet

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

Table 2-52 describes significant fields shown in Figure 2-92.

Table 2-52 Debug PPP Packet Field Descriptions

Field	Description
PPP	This is PPP debugging output.
Serial4	Interface number associated with this debugging information.
(o), O	This packet was detected as an output packet.
(i) I	This packet was detected as an input packet.
lcp_slqr()	Procedure name; running LQM, send a Link Quality Report (LQR).
lcp_rlqr()	Procedure name; running LQM, received an LQR.
input (C025)	The router received a packet of the specified packet type (in hex). A value of C025 indicates packet of type LQM.
state = OPEN	PPP state; normal state is OPEN.

Field	Description
magic = D21B4	Magic Number for indicated node; when output is indicated, this is the Magic Number of the node on which debugging is enabled. The actual Magic Number depends on whether the packet detected is indicated as I or O.
datagramsize = 52	Packet length including header.
code = ECHOREQ(9)	Code identifies the type of packet received. Both forms of the packet, string and hexadecimal, are presented.
len = 48	Packet length without header.
id = 3	ID number per Link Control Protocol (LCP) packet format.
pkt type 0xC025	Packet type in hexadecimal; typical packet types are C025 for LQM and C021 for LCP.
LCP ECHOREQ (9)	Echo Request; value in parentheses is the hexadecimal representation of the LCP type.
LCP ECHOREP (A)	Echo Reply; value in parentheses is the hexadecimal representation of the LCP type.

To elaborate on the displayed output, consider the partial exchange in Figure 2-93. This sequence shows that one side is using ECHO for its keepalives and the other side is using LQRs.

Figure 2-93 Partial Debug PPP Packet Output

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

The following discussion briefly outlines each line of this exchange.

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

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

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

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

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

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

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

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

Figure 2-94 shows sample **debug ppp negotiation** output. This is a normal negotiation, where both sides agree on network control program (NCP) parameters. In this case, protocol type IP is proposed and acknowledged.

Figure 2-94 Sample Debug PPP Negotiation Output

```
router# debug ppp negotiation

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

Table 2-53 describes significant fields shown in Figure 2-94.

Table 2-53 Debug PPP Negotiation Field Descriptions

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

The following discussion briefly outlines each line shown in the example provided in Figure 2-94.

The first two lines in Figure 2-94 indicate that the router is trying to bring up LCP and intends to use the indicated negotiation options (Quality Protocol and Magic Number). The value fields are the values of the options themselves. C025/3E8 translates to Quality Protocol LQM. 3E8 is the reporting period (in hundredths of a second). 3D56CAC is the value of the Magic Number for the router.

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

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

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

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

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

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

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

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

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

Figure 2-95 shows sample output when **debug ppp packet** and **debug ppp negotiation** output are enabled at the same time.

Figure 2-95 Sample Debug PPP Output with Packet and Negotiation Options Enabled

```
router# debug ppp negotiation
router# debug ppp packet

ppp: sending CONFREQ, type = 4 (CI_QUALITYTYPE), value = C025/3E8
ppp: sending CONFREQ, type = 5 (CI_MAGICNUMBER), value = D4C64
PPP Serial4: O LCP CONFREQ(1) id 4 (12) QUALITYTYPE (8) 192 37 0 0 3 232
   MAGICNUMBER (6) 0 13 76 100
PPP Serial4(i): pkt type 0xC021, datagramsize 22
PPP Serial4: I LCP CONFREQ(1) id 4 (12) QUALITYTYPE (8) 192 37 0 0 3 232
   MAGICNUMBER (6) 0 13 84 240
PPP Serial4: input(C021) state = REQSENT code = CONFREQ(1) id = 4 len = 18
ppp: received config for type = 4 (QUALITYTYPE) acked
ppp: received config for type = 5 (MAGICNUMBER) value = D54F0 acked
PPP Serial4: O LCP CONFACK(2) id 4 (12) QUALITYTYPE (8) 192 37 0 0 3 232
   MAGICNUMBER (6) 0 13 84 240 (ok)
PPP Serial4(i): pkt type 0xC021, datagramsize 22
PPP Serial4: I LCP CONFACK(2) id 4 (12) QUALITYTYPE (8) 192 37 0 0 3 232
   MAGICNUMBER (6) 0 13 76 100
PPP Serial4: input(C021) state = ACKSENT code = CONFACK(2) id = 4 len = 18
PPP Serial4: state = ACKSENT fsm_rconfack(C021): rcvd id 4
ppp: config ACK received, type = 4 (CI_QUALITYTYPE), value = C025
ppp: config ACK received, type = 5 (CI_MAGICNUMBER), value = D4C64
ipcp: sending CONFREQ, type = 3 (CI_ADDRESS), Address = 2.1.1.2
PPP Serial4: O IPCP CONFREQ(1) id 3 (10) Type3 (6) 2 1 1 2
PPP Serial4: I IPCP CONFREQ(1) id 3 (10) Type3 (6) 2 1 1 1
PPP Serial4(i): pkt type 0x8021, datagramsize 14
PPP Serial4: input(8021) state = REQSENT code = CONFREQ(1) id = 3 len = 10
ppp Serial4: Negotiate IP address: her address 2.1.1.1 (ACK)
ppp: ipcp_reqci: returning CONFACK.
PPP Serial4: O IPCP CONFACK(2) id 3 (10) Type3 (6) 2 1 1 1 (ok)
PPP Serial4: I IPCP CONFACK(2) id 3 (10) Type3 (6) 2 1 1 2
PPP Serial4: input(8021) state = ACKSENT code = CONFACK(2) id = 3 len = 10
PPP Serial4: state = ACKSENT fsm_rconfack(8021): rcvd id 3
ipcp: config ACK received, type = 3 (CI_ADDRESS), Address = 2.1.1.2
PPP Serial4(o): lcp_slqr() state = OPEN magic = D4C64, len = 48
PPP Serial4(i): pkt type 0xC025, datagramsize 52
PPP Serial4(i): lcp_rlqr() state = OPEN magic = D54F0, len = 48
PPP Serial4(i): pkt type 0xC025, datagramsize 52
PPP Serial4(i): lcp_rlqr() state = OPEN magic = D54F0, len = 48
PPP Serial4(o): lcp_slqr() state = OPEN magic = D4C64, len = 48
```

S2877

This field shows a decimal representation of the Magic Number.

This field shows a decimal representation of the NCP value.

This field shows a decimal representation of the reporting period.

This exchange represents a successful PPP negotiation for support of NCP type IPCP.

Figure 2-96 shows sample **debug ppp negotiation** output when the remote side of the connection is unable to respond to LQM requests.

Figure 2-96 Sample Debug PPP Negotiation Output When No Response Is Detected

```
router# debug ppp negotiation

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

Figure 2-97 shows sample output when no response is detected for configuration requests (with both **debug ppp negotiation** and **debug ppp packet** enabled).

Figure 2-97 Sample Debug PPP Output When No Response Is Detected (with Negotiation and Packet Enabled)

```
router# debug ppp negotiation
router# debug ppp packet

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

Figure 2-98 shows sample **debug ppp error** output. These messages might appear when the Quality Protocol option is enabled on an interface that is already running PPP.

Figure 2-98 Sample Debug PPP Error Output

```
router# debug ppp error

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

Table 2-54 describes significant fields shown in Figure 2-98.

Table 2-54 Debug PPP Error Field Descriptions

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

Figure 2-99 shows sample **debug ppp chap** output. When doing CHAP authentication, use this **debug** command to determine why an authentication fails. This command is also useful when doing PAP authentication.

Figure 2-99 Sample Debug PPP CHAP Output

```
router# debug ppp chap

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

In general, these messages are self-explanatory. Fields that appear in **debug ppp chap** displays that can show optional output are outlined in Table 2-55.

Table 2-55 Debug PPP CHAP Field Descriptions

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

debug qllc error

Use the **debug qllc error** EXEC command to display quality link line control (QLLC) errors. The **no** form of this command disables debugging output.

debug qllc error
no debug qllc error

Syntax Description

This command has no arguments or keywords.

Command Mode

EXEC

Usage Guidelines

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

Sample Display

Figure 2-100 shows sample **debug qllc error** output.

Figure 2-100 Sample Debug QLLC Error Output

```
router# debug qllc error

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

Explanations for individual lines of output from Figure 2-100 follow.

The following line indicates that the QLLC connection was closed:

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

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

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

debug qllc event

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

debug qllc event
no debug qllc event

Syntax Description

This command has no arguments or keywords.

Command Mode

EXEC

Usage Guidelines

Use the **debug qllc event** command to display primitives that might affect the state of a QLLC connection. An example of these events is the allocation of a QLLC structure for a logical channel indicator when an X.25 call has been accepted with the QLLC call user data. Other examples are the receipt and transmission of LAN explorer and XID frames.

Sample Display

Figure 2-101 shows sample **debug qllc event** output.

Figure 2-101 Sample Debug Qllc Event Output

```
router# debug qllc event

QLLC: allocating  new qllc lci 9
QLLC: tx POLLING TEST, da 4001.3745.1088, sa 4000.1111.0001
QLLC: rx explorer response, da 4000.1111.0001, sa c001.3745.1088, rif 08B0.1A91.1901.A040
QLLC: gen NULL XID, da c001.3745.1088, sa 4000.1111.0001, rif 0830.1A91.1901.A040, dsap
4, ssap 4
QLLC: rx XID response, da 4000.1111.0001, sa c001.3745.1088, rif 08B0.1A91.1901.A040
```

Explanations for representative lines of output in Figure 2-101 follow.

The following line indicates a new QLLC data structure has been allocated:

```
QLLC: allocating  new qllc lci 9
```

The following lines show transmission and receipt of LAN explorer or test frames:

```
QLLC: tx POLLING TEST, da 4001.3745.1088, sa 4000.1111.0001
QLLC: rx explorer response, da 4000.1111.0001, sa c001.3745.1088, rif 08B0.1A91.1901.A040
```

The following lines show XID events:

```
QLLC: gen NULL XID, da c001.3745.1088, sa 4000.1111.0001, rif 0830.1A91.1901.A040, dsap
4, ssap 4
QLLC: rx XID response, da 4000.1111.0001, sa c001.3745.1088, rif 08B0.1A91.1901.A040
```

debug qllc packet

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

debug qllc packet
no debug qllc packet

Syntax Description

This command has no arguments or keywords.

Command Mode

EXEC

Usage Guidelines

This command helps you to track down errors in the QLLC interactions with X.25 networks. The data shown by this command only flows through the router on the X25 connection. Use **debug qllc packet** in conjunction with **debug x25 all** to see the connection and the data that flows through the router.

Sample Display

Figure 2-102 shows sample **debug qllc packet** output.

Figure 2-102 Sample Debug QLLC Packet Output

```
router# debug qllc packet

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

Explanations for individual lines of output from Figure 2-102 follow.

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

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

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

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

debug qllc state

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

debug qllc state
no debug qllc state

Syntax Description

This command has no arguments or keywords.

Command Mode

EXEC

Usage Guidelines

Use the **debug qllc state** command to show when the state of a QLLC connection has changed. The typical QLLC connection goes from states ADM to SETUP to NORMAL. The NORMAL state indicates that a QLLC connection exists and is ready for data transfer.

Sample Display

Figure 2-103 shows sample **debug qllc state** output.

Figure 2-103 Sample Debug Qllc Event Output

```
router# debug qllc state

Serial2 QLLC O:  QSM-CMD
Serial2: X25 O D1 DATA (5) Q 8 lci 9 PS 4 PR 3
QLLC: state ADM -> SETUP
Serial2: X25 I D1 RR (3) 8 lci 9 PR 5
Serial2: X25 I D1 DATA (5) Q 8 lci 9 PS 3 PR 5
Serial2 QLLC I: QUA-RSPQLLC:  addr 00, ctl 73

QLLC: qsetupstate: recvd qua rsp
QLLC: state SETUP -> NORMAL
```

Explanations for representative lines of output in Figure 2-103 follow.

The following line indicates a QLLC connection attempt is changing state from ADM to SETUP:

```
QLLC: state ADM -> SETUP
```

The following line indicates a QLLC connection attempt is changing state from SETUP to NORMAL:

```
QLLC: state SETUP -> NORMAL
```

debug qllc timer

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

debug qllc timer
no debug qllc timer

Syntax Description

This command has no arguments or keywords.

Command Mode

EXEC

Usage Guidelines

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

Sample Display

Figure 2-104 shows sample **debug qllc timer** output.

Figure 2-104 Sample Debug QLLC Timer Output

```
router# debug qllc timer

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

Explanations for individual lines of output from Figure 2-104 follow.

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

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

Other messages are informational and appear every ten seconds.

debug qllc x25

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

debug qllc x25
no debug qllc x25

Syntax Description

This command has no arguments or keywords.

Command Mode

EXEC

Usage Guidelines

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

Sample Display

Figure 2-105 shows sample **debug qllc x25** output.

Figure 2-105 Sample Debug QLLC X25 Output

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

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

Table 2-56 describes fields of output that appear in Figure 2-105 follow.

Table 2-56 Debug QLLC X.25 Field Descriptions

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

debug rif

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

debug rif
no debug rif

Syntax Description

This command has no arguments or keywords.

Command Mode

EXEC

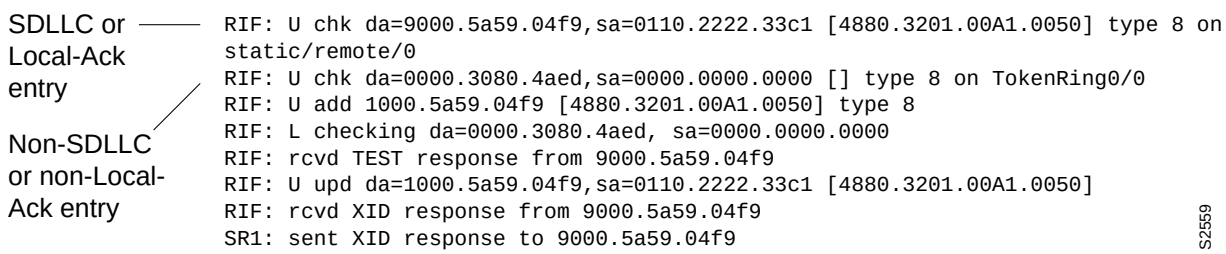
Usage Guidelines

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

Sample Display

Figure 2-106 shows sample **debug rif** output.

Figure 2-106 Sample Debug RIF Output



Explanations for representative lines of **debug rif** output in Figure 2-106 follow.

The first line of output is an example of a RIF entry for an interface configured for SDLLC or Local-Ack. Table 2-57 describes significant fields shown in this line of **debug rif** output.

Table 2-57 Debug RIF Field Descriptions—Part 1

Field	Description
RIF:	This message describes RIF debugging output.
U chk	Update checking. The entry is being updated; the timer is set to zero (0).
da = 9000.5a59.04f9	Destination MAC address.
sa = 0110.2222.33c1	Source MAC address. This field contains values of zero (0000.0000.0000) in a non-SDLLC or non-Local-ack entry.

Field	Description
[4880.3201.00A1.0050]	RIF string. This field is blank (null RIF) in a non-SDLLC or non-Local-Ack entry.
type 8	Possible values follow: 0—Null entry 1—This entry was learned from a particular Token Ring port (interface) 2—Statically configured 4—Statically configured for a remote interface 8—This entry is to be aged 16—This entry (which has been learned from a remote interface) is to be aged 32—This entry is not to be aged 64 —This interface is to be used by LAN Network Manager (and is not to be aged)
on static/remote/0	This route was learned from a real Token Ring port, in contrast to a virtual ring.

The following line of output is an example of a RIF entry for an interface that is not configured for SDLLC or Local-Ack:

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

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

The following line shows that a new entry has been added to the RIF cache:

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

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

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

The following line shows that a TEST response from address 9000.5a59.04f9 was inserted into the RIF cache:

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

The following line shows that the RIF entry for this route has been found and updated:

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

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

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

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

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

Table 2-58 explains the other possible lines of **debug rif** output.

Table 2-58 Debug RIF Field Descriptions—Part 2

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

debug sdlc

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

debug sdlc
no debug sdlc

Syntax Description

This command has no arguments or keywords.

Command Mode

EXEC

Usage Guidelines

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

Sample Display

Figure 2-107 shows sample **debug sdlc** output.

Figure 2-107 Sample Debug SDLC Output

```
router# debug sdlc

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

Explanations for individual lines of output from Figure 2-107 follow.

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

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

The following line of output describes a frame input event:

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

Table 2-59 describes the fields in this line of output.

Table 2-59 Debug SDLC Field Descriptions for a Frame Output Event

Field	Description
Serial3	Interface type and unit number reporting the frame event.
SDLC	Protocol providing the information.
O	Command mode of frame event. Possible values follow: I—Frame input O—Frame output T—T1 timer expired
(12495952)	Current timer value.
C2	SDLC address of the SDLC connection.
CONNECT	State of the protocol when the frame event occurred. Possible values follow: CONNECT DISCONNECT DISCSENT (disconnect sent) ERROR (FRMR frame sent) REJSENT (reject frame sent) SNRMSENT (SNRM frame sent) USBUSY THEMBUSY BOTHBUSY
(2)	Size of the frame (in bytes).
RR	Frame type name. Possible values follow: DISC—Disconnect DM—Disconnect mode FRMR—Frame reject IFRAME—Information frame REJ—Reject RNR—Receiver not ready RR—Receiver ready SIM—Set Initialization mode command SNRM—Set Normal Response Mode TEST—Test frame UA—Unnumbered acknowledgment XID—EXchange ID

Field	Description
P/F	Poll/Final bit indicator. Possible values follow: F—Final (printed for Response frames) P—Poll (printed for Command frames) P/F—Poll/Final (printed for RR, RNR and REJ frames, which can be either Command or Response frames)
6	Receive count; range: 0–7.

The following line of output describes a frame input event:

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

In addition to the fields described in Table 2-59, output for a frame input event also includes two additional fields, as described in Table 2-60.

Table 2-60 Debug SDLC Field Descriptions Unique to a Frame Input Event

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

The following line of output describes a frame timer event:

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

Table 2-61 describes the fields in this line of output.

Table 2-61 Debug SDLC Field Descriptions for a Timer Event

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

debug sdlc local-ack

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

debug sdlc local-ack [*number*]
no debug sdlc local-ack [*number*]

Syntax Description

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

This command has no arguments or keywords.

Command Mode

EXEC

Usage Guidelines

You can select the frame types you want to monitor; the frame types correspond to bit flags. You can select 1, 2, 4, or 7, which is the decimal value of the bit flag settings. If you select 1, the octet is set to 00000001. If you select 2, the octet is set to 0000010. If you select 4, the octet is set to 00000100. If you want to select all frame types, select 7; the octet is 00000111. The default is 7 for all events. Table 2-62 defines these bit flags.

Table 2-62 Debug SDLC Local-Ack Debugging Levels

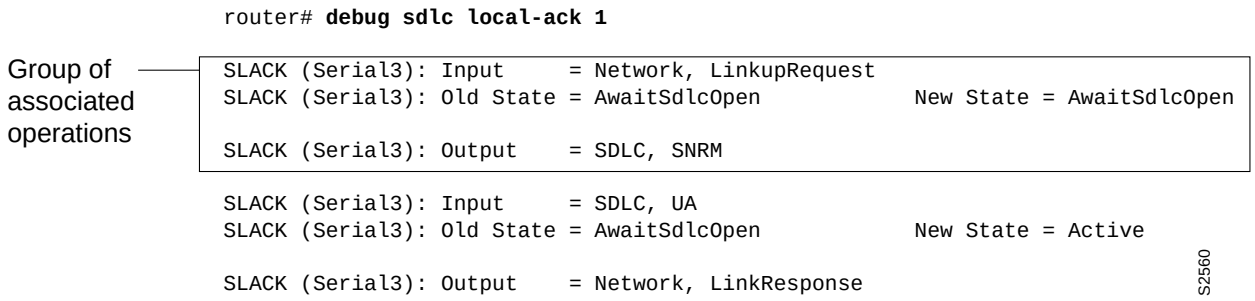
Debug Command	Meaning
debug sdlc local-ack 1	Only U-Frame events
debug sdlc local-ack 2	Only I-Frame events
debug sdlc local-ack 4	Only S-Frame events
debug sdlc local-ack 7	All SDLC Local-Ack events (default setting)

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

Sample Display

Figure 2-108 shows sample **debug sdlc local-ack** output.

Figure 2-108 Sample Debug SDLC Local-Ack Output



Explanations for individual lines of output from Figure 2-108 follow.

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

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

Table 2-63 describes the fields in this line of output.

Table 2-63 Debug SDLC Local-Ack Field Descriptions

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

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

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

The third line shows the output from the SDLC local acknowledgment state machine:

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

debug sdllc

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

debug sdllc
no debug sdllc

Syntax Description

This command has no arguments or keywords.

Command Mode

EXEC

Usage Guidelines

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

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

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

Sample Display

Figure 2-109 shows sample **debug sdllc** output between link layer peers from the perspective of the SDLLC-configured router.

Figure 2-109 Sample Debug SDLLC Output

```
router# debug sdllc

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

Table 2-64 describes significant fields shown in Figure 2-109:

Table 2-64 Debug SDLLC Field Descriptions

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

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

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

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

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

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

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

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

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

debug serial interface

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

debug serial interface
no debug serial interface

Syntax Description

This command has no arguments or keywords.

Command Mode

EXEC

Usage Guidelines

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

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

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

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

Debug Serial Interface for Frame Relay Encapsulation

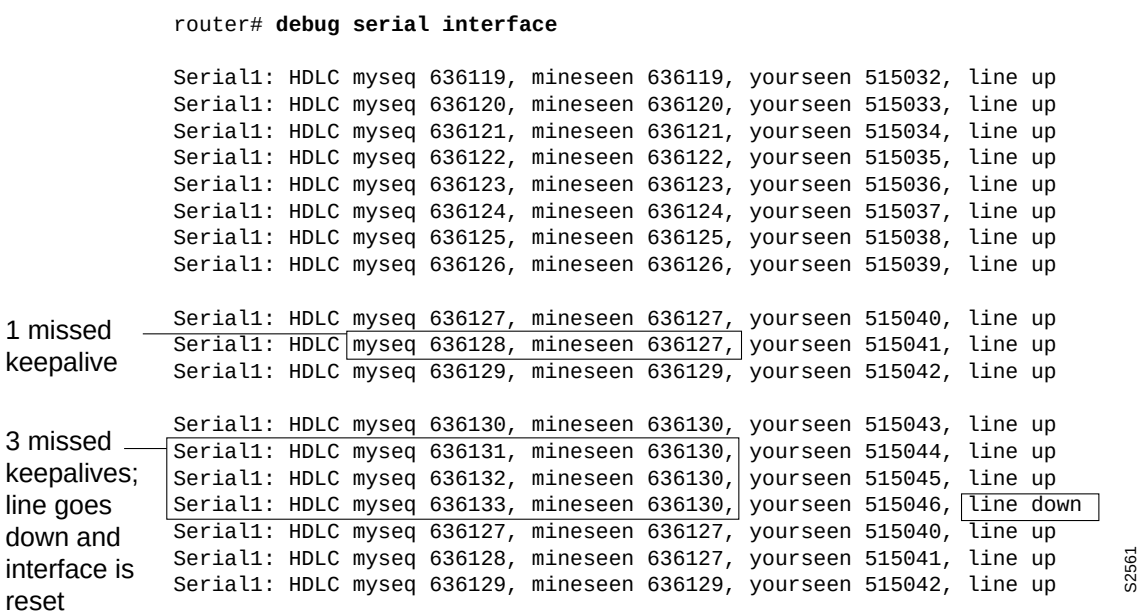
The following message is displayed if the encapsulation for the interface is Frame Relay (or HDLC) and the router attempts to send a packet containing an unknown packet type:

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

Debug Serial Interface for HDLC

Figure 2-110 shows sample **debug serial interface** output for an HDLC connection when keepalives are enabled.

Figure 2-110 Sample Debug Serial Interface Output for HDLC



In Figure 2-110, the **debug serial interface** display shows that the remote router is not receiving all the keepalives the router is sending. When the difference in the values in the myseq and mineseen fields exceeds three, the line goes down and the interface is reset.

Table 2-65 describes significant fields shown in Figure 2-110.

Table 2-65 Debug Serial Interface Field Descriptions for HDLC

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

Table 2-66 describes additional error messages that the **debug serial interface** command can generate for HDLC.

Table 2-66 Debug Serial Interface Error Messages for HDLC

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

Debug Serial Interface for HSSI

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

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

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

Debug Serial Interface for ISDN Basic Rate

Table 2-67 describes error messages that the **debug serial interface** command can generate for ISDN Basic Rate.

Table 2-67 Debug Serial Interface Message Descriptions for ISDN Basic Rate

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

Debug Serial Interface for an MK5025 Device

Table 2-68 describes the additional error messages that the **debug serial interface** command can generate for an MK5025 device.

Table 2-68 Debug Serial Interface Message Descriptions for an MK5025 Device

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

Debug Serial Interface for SMDS Encapsulation

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

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

The following message indicates that a new protocol requested SMDS to encapsulate the data for transmission. SMDS is not yet able to encapsulate the protocol.

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

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

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

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

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

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

```
SMDS: Invalid packet, Reserved NOT ZERO, x y
```

```
SMDS: Invalid packet, TAG mismatch x y
SMDS: Invalid packet, Bad TRAILER length x y
```

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

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

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

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

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

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

debug serial packet

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

debug serial packet
no debug serial packet

Syntax Description

This command has no arguments or keywords.

Command Mode

EXEC

Usage Guidelines

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

Sample Display

The **debug serial packet** command displays output for only SMDS encapsulations.

Debug Serial Packet for SMDS Encapsulation

Figure 2-111 shows sample output when SMDS is enabled on the interface.

Figure 2-111 Sample Debug Serial Packet Output for SMDS

```
router# debug serial packet

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

As Figure 2-111 shows, when encapsulation is set to SMDS, **debug serial packet** displays the entire SMDS header (in hex), as well as some payload data on transmit or receive. This information is useful only when you have an understanding of the SMDS protocol. The first line of the output indicates either Sending or Receiving.

debug source-bridge

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

debug source-bridge
no debug source-bridge

Syntax Description

This command has no arguments or keywords.

Command Mode

EXEC

Sample Display

Figure 2-112 shows sample **debug source-bridge** output for peer bridges using TCP as a transport mechanism. The remote source-route bridging (RSRB) network configuration has ring 2 and ring 1 bridged together through remote peer bridges. The remote peer bridges are connected via a serial line and use TCP as the transport mechanism.

Figure 2-112 Sample Debug Source-Bridge Output in TCP Environment

```
router# debug source-bridge

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

Explanations for individual lines of ouptut in Figure 2-112 follow.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Figure 2-113 shows sample **debug source-bridge** output for peer bridges using direct encapsulation as a transport mechanism. The RSRB network configuration has ring 1 and ring 2 bridged together through peer bridges. The peer bridges are connected via a serial line and use TCP as the transport mechanism.

Figure 2-113 Sample Debug Source-Bridge Output in Direct Encapsulation Environment

```
router# debug source-bridge

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

Explanations for individual lines of output in Figure 2-113 follow.

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

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

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

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

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

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

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

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

The following line indicates that bridge 1 and ring 1 were added to the source-bridge table for Serial1.

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

debug source event

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

debug source event
no debug source event

Syntax Description

This command has no arguments or keywords.

Command Mode

EXEC

Usage Guidelines

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

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

Sample Display

Figure 2-114 shows sample **debug source event** output.

Figure 2-114 Sample Debug Source Event Output

```
router# debug source event

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

Table 2-69 describes significant fields shown in Figure 2-114.

Table 2-69 Debug Source Event Field Descriptions

Field	Description
RSRB0:	Indication that this RIF cache entry is for the Token Ring 0 interface, which has been configured for remote source-route bridging. (SRB1, in contrast, would indicate that this RIF cache entry is for Token Ring 1, configured for source-route bridging.)

Field	Description
forward	Forward (normal data) packet, in contrast to a control packet containing proprietary Cisco bridging information.
srn 5	Ring number of the packet's source ring.
bn 1	Bridge number of the bridge this packet traverses.
trn 10	Ring number of the packet's target ring.
src: 8110.2222.33c1	Source address of the route in this RIF cache entry.
dst: 1000.5a59.04f9	Destination address of the route in this RIF cache entry.
[0800.3201.00A1.0050]	RIF string in this RIF cache entry.

Examples of other **debug source event** messages follow.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

The following example shows miscellaneous remote peer connection establishment messages:

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

debug span

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

debug span
no debug span

Syntax Description

This command has no arguments or keywords.

Command Mode

EXEC

Usage Guidelines

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

Sample Display—IEEE Spanning Tree

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

```
ST: Ether4 0000000000000A080002A02D6700000000000A080002A02D67800100000140002000F00
```

Figure 2-115 shows the preceding **debug span** output broken up by fields and labeled to aid documentation.

Figure 2-115 Sample Debug Span Output for an IEEE BPDU Packet

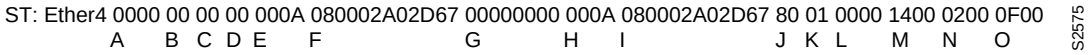


Table 2-70 describes significant fields shown in Figure 2-115.

Table 2-70 Debug Span Field Descriptions for an IEEE BPDU Packet

Field	Description
ST:	Indication that this is a spanning tree packet.
Ether4	Interface receiving the packet.
(A) 0000	Indication that this is an IEEE BPDU packet.
(B) 00	Version.
(C) 00	Command mode: 00 indicates config BPDU. 80 indicates the Topology Change Notification (TCN) BPDU.
(D) 00	Topology change acknowledgment: 00 indicates no change. 80 indicates a change notification.

Field	Description
(E) 000A	Root priority.
(F) 080002A02D67	Root ID.
(G) 00000000	Root path cost (0 means the sender of this BPDU packet is the root bridge).
(H) 000A	Bridge priority.
(I) 080002A02D67	Bridge ID.
(J) 80	Port priority.
(K) 01	Port No. 1.
(L) 0000	Message age in 256ths of a second (0 seconds, in this case).
(M) 1400	Maximum age in 256ths of a second (20 seconds, in this case).
(N) 0200	Hello time in 256ths of a second (2 seconds, in this case).
(O) 0F00	Forward delay in 256ths of a second (15 seconds, in this case).

Sample Display—DEC Spanning Tree

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

```
ST: Ethernet4 E1190100000200000C01A2C900640008000000C0106CE0A01050F1E6A
```

Figure 2-116 shows the preceding **debug span** output broken up by fields and labeled to aid documentation.

Figure 2-116 Sample Debug Span Output

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

S2576

Table 2-71 describes significant fields shown in Figure 2-116.

Table 2-71 Debug Span Field Descriptions for a DEC BPDU Packet

Field	Description
ST:	Indication that this is a spanning tree packet.
Ethernet4	Interface receiving the packet.
(A) E1	Indication that this is a DEC BPDU packet.
(B) 19	Indication that this is a DEC Hello packet. Possible values are as follows: 0x19—DEC Hello 0x02—Topology change notification (TCN)
(C) 01	DEC version.
(D) 00	Flag that is a bit field with the following mapping: 1—TCN 2—TCN acknowledgment 8—Use short timers
(E) 0002	Root priority.

Field	Description
(F) 00000C01A2C9	Root ID (MAC address).
(G) 0064	Root path cost (translated as 100 in decimal notation).
(H) 0080	Bridge priority.
(I) 00000C0106CE	Bridge ID.
(J) 0A	Port ID (in contrast to interface number).
(K) 01	Message age (in seconds).
(L) 05	Hello time (in seconds).
(M) 0F	Maximum age (in seconds).
(N) 1E	Forward delay (in seconds).
(O) 6A	Not applicable.

debug sse

Use the **debug sse** EXEC command to display information for the Silicon Switching Engine (SSE) processor. The **no** form of this command disables debugging output.

debug sse
no debug sse

Syntax Description

This command has no arguments or keywords.

Command Mode

EXEC

Usage Guidelines

By using the **debug sse** command, you can observe statistics and counters maintained by the SSE.

Sample Display

Figure 2-117 shows sample **debug sse** output.

Figure 2-117 Sample Debug SSE Output

```
router# debug sse
SSE: IP number of cache entries changed 273 274
SSE: IP number of cache entries changed 273 274
SSE: bridging enabled
SSE: interface Ethernet0/0 icb 0x30 addr 0x29 status 0x21A040 protos 0x11
SSE: interface Ethernet0/1 icb 0x33 addr 0x29 status 0x21A040 protos 0x11
SSE: interface Ethernet0/2 icb 0x36 addr 0x29 status 0x21A040 protos 0x10
SSE: interface Ethernet0/3 icb 0x39 addr 0x29 status 0x21A040 protos 0x11
SSE: interface Ethernet0/4 icb 0x3C addr 0x29 status 0x21A040 protos 0x10
SSE: interface Ethernet0/5 icb 0x3F addr 0x29 status 0x21A040 protos 0x11
SSE: interface Hssi1/0 icb 0x48 addr 0x122 status 0x421E080 protos 0x11
SSE: cache update took 316ms, elapsed 320ms
```

Explanations for representative lines of output in Figure 2-117 follow.

The following line indicates that the SSE cache is being updated due to a change in the IP fast switching cache:

```
SSE: IP number of cache entries changed 273 274
```

The following line indicates that bridging functions were enabled on the SSE:

```
SSE: bridging enabled
```

The following lines indicate that the SSE is now loaded with information about the interfaces:

```
SSE: interface Ethernet0/0 icb 0x30 addr 0x29 status 0x21A040 protos 0x11
SSE: interface Ethernet0/1 icb 0x33 addr 0x29 status 0x21A040 protos 0x11
SSE: interface Ethernet0/2 icb 0x36 addr 0x29 status 0x21A040 protos 0x10
SSE: interface Ethernet0/3 icb 0x39 addr 0x29 status 0x21A040 protos 0x11
SSE: interface Ethernet0/4 icb 0x3C addr 0x29 status 0x21A040 protos 0x10
SSE: interface Ethernet0/5 icb 0x3F addr 0x29 status 0x21A040 protos 0x11
SSE: interface Hssi1/0 icb 0x48 addr 0x122 status 0x421E080 protos 0x11
```

The following line indicates that the SSE took 316 ms of processor time to update the SSE cache. The value of 320 ms represents the total time elapsed while the cache updates were performed.

```
SSE: cache update took 316ms, elapsed 320ms
```

debug standby

Use the **debug standby** EXEC command to display hot standby protocol state changes. The **no** form of this command disables debugging output.

debug standby
no debug standby

Syntax Description

This command has no arguments or keywords.

Command Mode

EXEC

Usage Guidelines

The **debug standby** command displays hot standby protocol state changes and debugging information regarding transmission and receipt of hot standby protocol packets. Use this command to determine whether hot standby routers recognize one another and take the proper actions.

Sample Display

Figure 2-118 shows sample **debug standby** output.

Figure 2-118 Sample Debug Standby Output

```
router# debug standby

SB: Ethernet0 state Virgin -> Listen
SB: Starting up hot standby process
SB:Ethernet0 Hello in 198.92.72.21 Active pri 90 hel 3 hol 10 ip 198.92.72.29
SB:Ethernet0 Hello in 198.92.72.21 Active pri 90 hel 3 hol 10 ip 198.92.72.29
SB:Ethernet0 Hello in 198.92.72.21 Active pri 90 hel 3 hol 10 ip 198.92.72.29
SB:Ethernet0 Hello in 198.92.72.21 Active pri 90 hel 3 hol 10 ip 198.92.72.29
SB: Ethernet0 state Listen -> Speak
SB:Ethernet0 Hello out 198.92.72.20 Speak pri 100 hel 3 hol 10 ip 198.92.72.29
SB:Ethernet0 Hello in 198.92.72.21 Active pri 90 hel 3 hol 10 ip 198.92.72.29
SB:Ethernet0 Hello out 198.92.72.20 Speak pri 100 hel 3 hol 10 ip 198.92.72.29
SB:Ethernet0 Hello in 198.92.72.21 Active pri 90 hel 3 hol 10 ip 198.92.72.29
SB:Ethernet0 Hello out 198.92.72.20 Speak pri 100 hel 3 hol 10 ip 198.92.72.29
SB:Ethernet0 Hello in 198.92.72.21 Active pri 90 hel 3 hol 10 ip 198.92.72.29
SB: Ethernet0 state Speak -> Standby
SB:Ethernet0 Hello out 198.92.72.20 Standby pri 100 hel 3 hol 10 ip 198.92.72.29
SB:Ethernet0 Hello in 198.92.72.21 Active pri 90 hel 3 hol 10 ip 198.92.72.29
SB:Ethernet0 Hello out 198.92.72.20 Standby pri 100 hel 3 hol 10 ip 198.92.72.29
SB:Ethernet0 Hello in 198.92.72.21 Active pri 90 hel 3 hol 10 ip 198.92.72.29
SB:Ethernet0 Hello out 198.92.72.20 Standby pri 100 hel 3 hol 10 ip 198.92.72.29
SB:Ethernet0 Hello in 198.92.72.21 Active pri 90 hel 3 hol 10 ip 198.92.72.29
SB: Ethernet0 Coup out 198.92.72.20 Standby pri 100 hel 3 hol 10 ip 198.92.72.29
SB: Ethernet0 state Standby -> Active
SB:Ethernet0 Hello out 198.92.72.20 Active pri 100 hel 3 hol 10 ip 198.92.72.29
SB:Ethernet0 Hello in 198.92.72.21 Speak pri 90 hel 3 hol 10 ip 198.92.72.29
SB:Ethernet0 Hello out 198.92.72.20 Active pri 100 hel 3 hol 10 ip 198.92.72.29
SB:Ethernet0 Hello in 198.92.72.21 Speak pri 90 hel 3 hol 10 ip 198.92.72.29
SB:Ethernet0 Hello out 198.92.72.20 Active pri 100 hel 3 hol 10 ip 198.92.72.29
```

Table 2-72 describes significant fields shown in Figure 2-118.

Table 2-72 Debug Standby Field Descriptions

Field	Description
SB	An abbreviation for “standby.”
Ethernet0	The interface on which a hot standby packet was sent or received.
Hello in	Hello packet received from the specified IP address.
Hello out	Hello packet sent from the specified IP address.
pri	Priority advertised in the hello packet.
hel	Hello interval advertised in the hello packet.
hol	Holddown interval advertised in the hello packet.
ip *address*	Hot standby group IP address advertised in the hello packet.
state	Transition from one state to another.
Coup out *address*	Coup packet sent by the router from the specified IP address.

Explanations for representative lines of output in Figure 2-118 follow.

The following line indicates that the router is initiating the hot standby protocol. The **standby ip** interface configuration command enables hot standby.

```
SB: Starting up hot standby process
```

The following line indicates that a state transition occurred on the interface:

```
SB: Ethernet0 state Listen -> Speak
```

debug stun packet

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

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

Syntax Description

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

Command Mode

EXEC

Usage Guidelines

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

Sample Display

Figure 2-119 shows sample **debug stun packet** output.

Figure 2-119 Sample Debug STUN Packet Output

```
                      router# debug stun packet
X1 type
of packet         —   STUN sdlc: 0:00:04 Serial3        NDI: (0C2/008) U: SNRM    PF:1
                      STUN sdlc: 0:00:04 Serial3        NDI: (0C2/008) U: SNRM    PF:1
                      STUN sdlc: 0:00:01 Serial3        SDI: (0C2/008) U: UA      PF:1
X2 type           —   STUN sdlc: 0:00:00 Serial3        SDI: (0C2/008) S: RR      PF:1  NR:000
of packet             STUN sdlc: 0:00:00 Serial3        SDI: (0C2/008) S: RR      PF:1  NR:000
                      STUN sdlc: 0:00:00 Serial3        SDI: (0C2/008) S: RR      PF:1  NR:000
                      STUN sdlc: 0:00:00 Serial3        SDI: (0C2/008) S: RR      PF:1  NR:000
                      STUN sdlc: 0:00:00 Serial3        SDI: (0C2/008) S: RR      PF:1  NR:000
                      STUN sdlc: 0:00:00 Serial3        SDI: (0C2/008) S: RR      PF:1  NR:000
                      STUN sdlc: 0:00:00 Serial3        SDI: (0C2/008) S: RR      PF:1  NR:000
                      STUN sdlc: 0:00:00 Serial3        SDI: (0C2/008) S: RR      PF:1  NR:000
                      STUN sdlc: 0:00:00 Serial3        SDI: (0C2/008) S: RR      PF:1  NR:000
                      STUN sdlc: 0:00:00 Serial3        SDI: (0C2/008) S: RR      PF:1  NR:000
                      STUN sdlc: 0:00:00 Serial3        SDI: (0C2/008) S: RR      PF:1  NR:000
X3 type           —   STUN sdlc: 0:00:00 Serial3        NDI: (0C2/008) I:         PF:1  NR:000  NS:000
of packet             STUN sdlc: 0:00:00 Serial3        SDI: (0C2/008) I:         PF:1  NR:001  NS:000
                      STUN sdlc: 0:00:00 Serial3        SDI: (0C2/008) S: RR      PF:1  NR:001
                      STUN sdlc: 0:00:00 Serial3        SDI: (0C2/008) S: RR      PF:1  NR:001
                      STUN sdlc: 0:00:00 Serial3        SDI: (0C2/008) S: RR      PF:1  NR:001
                      STUN sdlc: 0:00:00 Serial3        SDI: (0C2/008) S: RR      PF:1  NR:001
```

S2563

Explanations for individual lines of output from Figure 2-119 follow.

The following line describes an X1 type of packet:

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

Table 2-73 describes significant fields shown in this line of **debug stun packet** output.

Table 2-73 Debug STUN Packet Field Descriptions

Field	Description
STUN sdlc:	Indication that the STUN feature is providing the information.
0:00:04	Time elapsed since receipt of previous packet.
Serial3	Interface type and unit number reporting the event.
NDI:	The type of cloud separating the SDLC end nodes. Possible values follow: NDI—Network input SDI—Serial link
0C2	SDLC address of the SDLC connection.
008	A modulo value of 8.

Field	Description
U:SNRM	The frame type followed by the command or response type. In this case it is an Unnumbered frame that contains an SNRM (Set Normal Response Mode) command. The possible frame types are as follows: I—Information frame S—Supervisory frame. The possible commands and responses are: RR (Receive Ready), RNR (Receive Not Ready), and REJ (Reject). U—Unnumbered frame. The possible commands are: UI (Unnumbered Information), SNRM, DISC/RD (Disconnect/Request Disconnect), SIM/RIM, XID Exchange Identification), TEST. The possible responses are UA (unnumbered acknowledgment), DM (Disconnected Mode), and FRMR (Frame Reject Mode)
PF:1	Poll/Final bit. 0—Off 1—On

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

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

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

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

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

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

debug tftp

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

debug tftp
no debug tftp

Syntax Description

This command has no arguments or keywords.

Command Mode

EXEC

Sample Display

Figure 2-120 shows sample **debug tftp** output from the EXEC command **write network**.

Figure 2-120 Sample Debug TFTP Output

```
router# debug tftp

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

Table 2-74 describes significant fields shown in the first line of output from Figure 2-120.

Table 2-74 Debug TFTP Field Descriptions

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

debug token ring

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

debug token ring
no debug token ring

Syntax Description

This command has no arguments or keywords.

Command Mode

EXEC

Usage Guidelines

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

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

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

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

Sample Display

Figure 2-121 shows sample **debug token ring** output.

Figure 2-121 Sample Debug Token Ring Output

```
router# debug token ring

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

Table 2-75 describes significant fields shown in the second line of output from Figure 2-121.

Table 2-75 Debug Token Ring Field Descriptions—Part 1

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

Table 2-76 describes significant fields shown in the third line of output from Figure 2-121.

Table 2-76 Debug Token Ring Field Descriptions—Part 2

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

Table 2-77 describes significant fields shown in the fifth line of output from Figure 2-121.

Table 2-77 Debug Token Ring Field Descriptions—Part 3

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

debug vines arp

Use the **debug vines arp** EXEC command to display debugging information on all Virtual Integrated Network Service (VINES) Address Resolution Protocol (ARP) packets that the router sends or receives. The **no** form of this command disables debugging output.

debug vines arp
no debug vines arp

Syntax Description

This command has no arguments or keywords.

Command Mode

EXEC

Sample Display

Figure 2-122 shows sample **debug vines arp** output.

Figure 2-122 Sample Debug VINES ARP Output

```
router# debug vines arp

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

In Figure 2-122, the first four lines show a non-sequenced ARP transaction and the second four lines show a sequenced ARP transaction. Within the first group of four lines, the first line shows that the router received an ARP request (type 0) from indicated station address 0260.8c43.a7e4. The second line shows that the router is sending back the ARP service response (type 1), indicating that it is willing to assign VINES Internet addresses. The third line shows that the router received a VINES Internet address assignment request (type 2) from address 0260.8c43.a7e4. The fourth line shows that the router is responding (type 3) to the address assignment request from the client and assigning it the address 3001153C:8004.

Within the second group of four lines, the sequenced ARP packet also includes the router' current sequence number and the metric value between the router and the client.

Table 2-78 describes significant fields shown in Figure 2-122.

Table 2-78 Debug VINES ARP Field Descriptions

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

debug vines echo

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

debug vines echo
no debug vines echo

Syntax Description

This command has no arguments or keywords.

Note These echo packets do not include network layer addresses.

Command Mode

EXEC

Sample Display

Figure 2-123 shows sample **debug vines echo** output.

Figure 2-123 Sample Debug VINES Echo Output

```
router# debug vines echo

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

Table 2-79 describes the fields shown in Figure 2-123.

Table 2-79 Debug VINES Echo Field Descriptions

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

debug vines ipc

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

debug vines ipc
no debug vines ipc

Syntax Description

This command has no arguments or keywords.

Command Mode

EXEC

Usage Guidelines

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

Sample Display

Figure 2-124 shows sample **debug vines ipc** output for three pairs of transactions. For more information about these fields or their values, refer to Banyan VINES documentation.

Figure 2-124 Sample Debug VINES IPC Output

```
router# debug vines ipc

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

Table 2-80 describes the fields shown in Figure 2-124.

Table 2-80 VINES IPC Field Descriptions

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

debug vines netrpc

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

debug vines netrpc
no debug vines netrpc

Syntax Description

This command has no arguments or keywords.

Command Mode

EXEC

Usage Guidelines

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

Sample Display

Figure 2-125 shows sample **debug vines netrpc** output. For more information about these fields or their values, refer to Banyan VINES documentation.

Figure 2-125 Sample Debug VINES NetRPC Output

```
router# debug vines netrpc

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

Table 2-81 describes the fields shown in the first line of output in Figure 2-125.

Table 2-81 Debug VINES NetRPC Field Descriptions

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

debug vines packet

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

debug vines packet
no debug vines packet

Syntax Description

This command has no arguments or keywords.

Command Mode

EXEC

Sample Display

Figure 2-126 shows sample **debug vines packet** output.

Figure 2-126 Sample Debug VINES Packet Output

```
router# debug vines packet

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

The following information describes selected lines of output from Figure 2-126.

Table 2-82 describes the fields shown in the first line of output.

Table 2-82 Debug VINES Packet Field Descriptions

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

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

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

In the following line, the router being debugged is the destination address (3000B959:1):

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

In the following line, (local) indicates that the router being debugged generated the packet:

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

debug vines routing

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

debug vines routing [**verbose**]
no debug vines routing

Syntax Description

verbose	(Optional) Provides detailed information about the contents of each update.

Command Mode

EXEC

Sample Displays

Figure 2-127 shows sample **debug vines routing** output.

Figure 2-127 Sample Debug VINES Routing Output

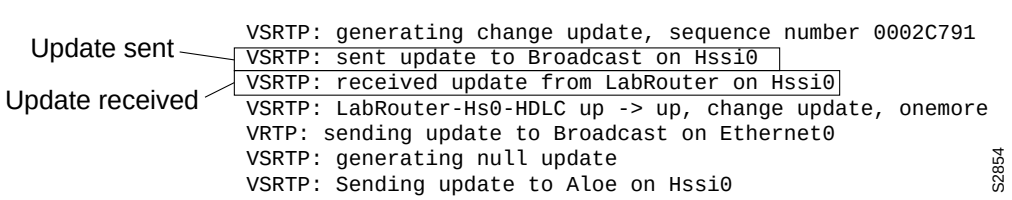


Figure 2-128 shows sample **debug vines routing verbose** output.

Figure 2-128 Sample Debug VINES Routing Verbose Output

```
router# debug vines routing verbose

VRTP: sending update to Broadcast on Ethernet0
    network 30011E7E, metric 0020 (0.4000 seconds)
    network 30015800, metric 0010 (0.2000 seconds)
    network 3003148A, metric 0020 (0.4000 seconds)
VSRTP: generating change update, sequence number 0002C795
    network Router9        metric 0010, seq 00000000, flags 09
    network RouterZZ       metric 0230, seq 00052194, flags 02
VSRTP: sent update to Broadcast on Hssi0
VSRTP: received update from LabRouter on Hssi0
    update: type 00, flags 07, id 000E, ofst 0000, seq 15DFC, met 0010
    network LabRouter from the server
    network Router9        metric 0020, seq 00000000, flags 09
VSRTP: LabRouter-Hs0-HDLC up -> up, change update, onemore
```

Figure 2-128 describes two VINES routing updates; the first includes two entries and the second includes three entries. The following information describes selected lines of output.

The following line shows that the router sent a periodic routing update to the broadcast address FFFFFFFF:FFFF through the Ethernet0 interface:

```
VRTP: sending update to Broadcast on Ethernet0
```

The following line indicates that the router knows how to reach network 30011E7E, which is a metric of 0020 away from the router. The value that follows the metric (0.4000 seconds) interprets the metric in seconds.

```
network 30011E7E, metric 0020 (0.4000 seconds)
```

The following lines show that the router sent a change routing update to the Broadcast addresses on the Hssi0 interface using the Sequenced Routing Update Protocol (SRTP) routing protocol:

```
VSRTP: generating change update, sequence number 0002C795
VSRTP: Sending update to Broadcast on Hssi0
```

The lines in between the previous two indicate that the router knows how to reach network Router9, which is a metric of 0010 (0.2000 seconds) away from the router. The sequence number for Router9 is zero, and according to the 0x08 bit in the flags field, is invalid. The 0x01 bit of the flags field indicates that Router9 is attached via a LAN interface.

```
network Router9        metric 0010, seq 00000000, flags 09
```

The next lines indicate that the router can reach network RouterZZ, which is a metric of 0230 (7.0000 seconds) away from the router. The sequence number for RouterZZ is 0052194. The 0x02 bit of the flags field indicates that RouterZZ is attached via a WAN interface.

```
network RouterZZ       metric 0230, seq 00052194, flags 02
```

The following line indicates that the router received a routing update from the router LabRouter through the Hssi0 interface:

```
VINESRTP: received update from LabRouter on Hssi0
```

The following line displays all SRTP values contained in the header of the SRTP packet. This is a type 00 packet, which is a routing update, and the flags field is set to 07, indicating that this is a change update (0x04) and contains both the beginning (0x01) and end (0x02) of the update. This overall update is update number 000E from the router, and this fragment of the update contains the routes beginning at offset 0000 of the update. The sending router's sequence number is currently 00015DFC, and its configured metric for this interface is 0010.

```
update: type 00, flags 07, id 000E, ofst 0000, seq 00015DFC, met 0010
```

The following line implies that the server sending this update is directly accessible to the router (even though VINES servers do not explicitly list themselves in routing updates). Because this is an implicit entry in the table, the other information for this entry is taken from the previous line.

```
network LabRouter from the server
```

As the first actual entry in the routing update from LabRouter, the following line indicates that Router9 can be reached by sending to this server. This network is a metric of 0020 away from the sending server.

```
network Router9        metric 0020, seq 00000000, flags 09
```

debug vines service

Use the **debug vines service** EXEC command to display information on all transactions that occur at the VINES Service (or applications) layer. The **no** form of this command disables debugging output.

debug vines service
no debug vines service

Syntax Description

This command has no arguments or keywords.

Command Mode

EXEC

Usage Guidelines

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

Note Because the **debug vines service** command provides the highest level overview of VINES traffic through the router, it is best to begin debugging using this command, and then proceed to use lower-level VINES **debug** commands as necessary.

Sample Display

Figure 2-129 shows sample **debug vines service** output.

Figure 2-129 Sample Debug VINES Service Output

```
router# debug vines service

VSRV: Get Time Info sent to Townsaver
VSRV: Get Time Info response from Townsaver, time: 01:47:54 PDT Apr 29 1993
VSRV:     epoch SS@Aloe@Servers-10, age: 0:15:15
```

Sent/ Response pair

S2565

As Figure 2-129 suggests, **debug vines service** lines of output appear as activity pairs—either a sent/response pair as shown, or as a received/sent pair.

Table 2-83 describes the fields shown in the second line of output in Figure 2-129. For more information about these fields or their values, refer to Banyan VINES documentation.

Table 2-83 Debug VINES Service Field Descriptions—Part 1

Field	Description
VSRV:	Indicates that this is output from the **debug vines service** command.
Get Time Info	Indicates one of three packet types: Get Time Info Time Set Time Sync
response from	Indicates whether the packet was sent to another router, a response from another router, or received from another router.
Townsaver	Indicates the machine name as assigned using the VINES **host** command, or IP address of the other router.
time: 01:47:54 PDT Apr 29 1993	Indicates the current time in hours:minutes:seconds and current date.

Table 2-84 describes the fields shown in the third line of output in Figure 2-129. This line is an extension of the first two lines of output. For more information about these fields or their values, refer to Banyan VINES documentation.

Table 2-84 Debug VINES Service Field Descriptions—Part 2

Field	Description
VSRV:	Output from the **debug vines service** command.
epoch	Line of output that describes a VINES epoch.
SS@Aloe@Servers-10	Epoch name.
age: 0:15:15	Epoch—elapsed time since the time was last set in the network.

debug vines state

Use the **debug vines state** EXEC command to display information on the VINES SRTP state machine transactions. The **no** form of this command disables debugging output.

debug vines state
no debug vines state

Syntax Description

This command has no arguments or keywords.

Command Mode

EXEC

Usage Guidelines

This command provides a subset of the information provided by the **debug vines routing** command, showing only the transactions made by the SRTP state machine. Refer to the **debug vines routing** command for descriptions of output from the **debug vines state** command.

debug vines table

Use the **debug vines table** EXEC command to display information on all modifications to the VINES routing table. The **no** form of this command disables debugging output.

debug vines table
no debug vines table

Syntax Description

This command has no arguments or keywords.

Command Mode

EXEC

Usage Guidelines

This command provides a subset of the information produced by the **debug vines routing** command, as well as some more detailed information on table additions and deletions.

Sample Display

Figure 2-130 shows sample **debug vines table** output.

Figure 2-130 Sample Debug VINES Table Output

```
router# debug vines table

VINESRTP: create neighbor 3001153C:8004, interface Ethernet0
```

Table 2-85 describes significant fields shown in Figure 2-130.

Table 2-85 Debug VINES Table Field Descriptions

Field	Description
VINESRTP:	Indicates that this is a **debug vines routing** or **debug vines table** message.
create neighbor 3001153C:8004	Indicates that the client at address 3001153C:8004 has been added to the Banyan VINES neighbor table.
interface Ethernet 0	Indicates that this neighbor can be reached through the router interface named Ethernet0.

debug x25 all

Use the **debug x25 all** EXEC command to display information on all X.25 traffic, including data, control messages, and flow control (RR and RNR) packets. The **no** form of this command disables debugging output.

debug x25 all
no debug x25 all

Syntax Description

This command has no arguments or keywords.

Command Mode

EXEC

Usage Guidelines

This command is particularly useful for diagnosing problems encountered when placing calls.

The **debug x25 all** output includes data, control messages, and flow control packets for all of the router's virtual circuits. The **debug x25 events** and **debug x25 vc** commands provide a subset of this output.

Caution Because **debug x25 all** displays all X.25 traffic, it is processor intensive and can render the router useless. Only use **debug x25 all** when the aggregate of all X.25 traffic is fewer than five packets per second.

Sample Display

Figure 2-131 shows sample **debug x25 all** output.

Figure 2-131 Sample Debug X25 All Output

```
router# debug x25 all

Serial2: X25 O R3 RESTART (5) 8 lci 0 cause 7 diag 0
Serial2: X25 I R3 RESTART (5) 8 lci 0 cause 0 diag 0
Serial2: X25 I P1 CALL REQUEST (11) 8 lci 1024
From (2): 49 To(2): 46
  Facilities: (0)
  Call User Data (4): 0xCC 00 00 00 (ip)
Serial2: X25 O P4 CALL CONNECTED (3) 8 lci 1024
Serial2: X25 I P4 DATA (103) 8 lci 1024 PS 0 PR 0
Serial2: X25 O D1 DATA (103) 8 lci 1024 PS 0 PR 1
Serial2: X25 I D1 DATA (103) 8 lci 1024 PS 1 PR 0
Serial2: X25 O D1 DATA (103) 8 lci 1024 PS 1 PR 2
Serial2: X25 I D1 RR (3) 8 lci 1024 PR 2
Serial2: X25 I D1 DATA (103) 8 lci 1024 PS 2 PR 2
Serial2: X25 O D1 DATA (103) 8 lci 1024 PS 2 PR 3
Serial2: X25 I D1 CLEAR REQUEST (5) 8 lci 1024 cause 0 diag 122
Serial2: X25 O D1 CLEAR CONFIRMATION (3) 8 lci 1024
XOT: X25 O D1 PVC-SETUP, waiting to connect (29) <Serial2 pvc 3><Serial2 pvc 1> 2/1 128/64
XOT: X25 I D1 PVC-SETUP, connected (29) <Serial2 pvc 3><Serial2 pvc 1> 2/1 128/64
Serial2: X25 O D1 RESET REQUEST (5) 8 lci 3 cause 15 diag 0
Serial2: X25 I D1 RESET CONFIRMATION (3) 8 lci 3
```

Figure 2-131 shows a typical exchange of packets between two X.25 devices on a network. The first line of output in Figure 2-131 describes a RESTART packet. Table 2-86 describes the fields in this line of output.

Table 2-86 Debug X25 All Field Descriptions

Field	Description
Serial2	The interface on which the X.25 event occured. Events that occur on an X.25-over-TCP connection report XOT.
X25	That this message describes an X.25 event.
O	Indication of whether the X.25 message was input (I) or output (O) through the interface.
R3	State of the virtual circuit. Possible values follow: D1—Flow control ready D2—DTE reset request D3—DCE reset indication P1—Idle P2—DTE waiting for DCE to connect CALL P3—DCE waiting for DTE to accept CALL P4—Data transfer P5—CALL collision

Field	Description
R3 (Continued)	P6—DTE clear request P7—DCE clear indication R1—Packet level ready R2—DTE restart request R3—DCE restart indication X1—Nonstandard state for a virtual circuit in hold-down See Annex B of the 1984 ITU-T X.25 Recommendation for more information on these states.
RESTART	The type of X.25 packet. Possible values follow: CALL CONNECTED CALL REQUEST CLEAR CONFIRMATION CLEAR REQUEST DATA DIAGNOSTIC ILLEGAL INTR CONFIRMATION INTR (interrupt) PVC-SETUP REGISTRATION REGISTRATION CONFIRMATION RESET CONFIRMATION RESET REQUEST RESTART RESTART CONFIRMATION RNR (Receiver Not Ready) RR (Receiver Ready)
(5)	Number of bytes in the packet.
8	Modulo of the virtual circuit. Possible values are 8 or 128.
lci 0	Virtual circuit number. See Annex A of the 1984 ITU-T X.25 Recommendation for information on VC assignment.
cause 7	Code indicating the event that triggered the packet. The cause field can only appear in entries for CLEAR REQUEST, RESET REQUEST, and RESTART packets. Possible values for the cause field can vary, depending on the type of packet. Refer to the “X.25 Cause and Diagnostic Codes” appendix for explanations of these codes.
diag 0	Code providing an additional hint as to what, if anything, went wrong. The diag field can only appear in entries for CLEAR REQUEST, DIAGNOSTIC (as “error 0”), RESET REQUEST and RESTART packets. Because of the large number of possible values, they are listed in the “X.25 Cause and Diagnostic Codes” appendix.

Table 2-87 describes the PS and PR fields that can appear in a **debug x25 all** display.

Table 2-87 Debug X25 All PS and PR Field Descriptions

Field	Description
PS 0	Packet send sequence number; used for flow control of the outgoing packet stream. Present only in DATA packets.
PR 0	Packet receive sequence number used for flow control of the incoming packet stream by indicating the PS value that the sender next expects to see.

In Figure 2-131, notice also that the CALL REQUEST packet precedes three other lines of output that have a unique format.

```
Serial2: X25 I P1 CALL REQUEST (11) 8 lci 1024
From (2): 49 To(2): 46
  Facilities: (0)
  Call User Data (4): 0xCC 00 00 00 (ip)
Serial2: X25 O P4 CALL CONNECTED (3) 8 lci 1024
```

These lines indicate that the CALL REQUEST packet has a two-digit source address, 49, and a two-digit destination address, 46. These are X.121 addresses that can be from 0 to 15 digits in length. The Facilities field is (0) bytes in length, indicating that no X.25 facilities are being requested. The optional call user data field is 4 bytes in length. Any encapsulation protocol identification (PID) in the Call User Data will have the encoding values printed and identified. Multiprotocol Virtual Circuits can also have PID information in Data packets; the debug output for these packets will also describe the PID.

The two lines of output in Figure 2-131 that begin with XOT are shown below.

```
XOT: X25 O D1 PVC-SETUP, waiting to connect (29) <Serial2 pvc 3><Serial2 pvc 1> 2/1 128/64
XOT: X25 I D1 PVC-SETUP, connected (29) <Serial2 pvc 3><Serial2 pvc 1> 2/1 128/64
```

These lines of output do not describe standard X.25 packets. Instead, they describe messages that represent a tunneled PVC setup between two routers. Table 2-88 describes the fields these two lines of output.

Table 2-88 Debug X25 All Field Descriptions for Packets Representing Tunneled PVC Activity

Field	Description
XOT	This message travels over a TCP connection.
X25	This message describes an X.25 event.
O	Indication of whether the X.25 message was input (I) or output (O) through the connection.
D1	State of the permanent virtual circuit. Possible values follow. D1—Flow control ready D2—DTE reset request D3—DCE reset indication See Annex B of the 1984 ITU-T X.25 Recommendation for more information on these states.

Field	Description
wait to connect	State of the PVC. Some of these strings only apply to PVCs that are remotely tunneled over a TCP connection. The %X25-3-PVCBAD system error message (as documented in the *System Error Messages* publication), and the **show x25 vc** command (as documented in the *Router Products Command Reference* publication) also use these PVC state strings. Possible values follow: awaiting PVC-SETUP reply can't support flow control values connected dest. disconnected dest. interface is not up dest. PVC configuration mismatch mismatched flow control values no such dest. interface no such dest. PVC non-X.25 dest. interface PVC setup protocol error PVC/TCP connect timed out PVC/TCP connection refused PVC/TCP routing error trying to connect via TCP waiting to connect
(29)	Incoming/outgoing message size (in bytes).
<Serial2 pvc 3>	Interface and PVC number that originated the message (originator).
<Serial2 pvc 1>	Interface and PVC number that responded to that message (responder).
2/1	Window sizes (in packets).
128/64	Maximum packet sizes (in bytes).

debug x25 events

Use the **debug x25 events** EXEC command to display information on all X.25 traffic except X.25 data or acknowledgment packets. The **no** form of this command disables debugging output.

debug x25 events
no debug x25 events

Syntax Description

This command has no arguments or keywords.

Command Mode

EXEC

Usage Guidelines

The **debug x25 events** command is useful for debugging X.25 problems, because it shows changes that occur in the virtual circuits handled by the router. Because most X.25 connectivity problems stem from errors that CLEAR or RESET virtual circuits, you can use **debug x25 events** to identify these errors.

While **debug x25 all** output includes both data and control messages for all of the router's virtual circuits, **debug x25 events** output includes only control messages for all of the router's VCs. In contrast, **debug x25 vc** output filters the output for a single VC number. Thus, **debug x25 events** output is a subset of **debug x25 all** output, and **debug x25 vc** output modifies either of them to further limit the output.

Note Because **debug x25 events** displays a subset of all X.25 traffic, it is safer to use than **debug x25 all** during production hours.

Sample Display

Figure 2-132 shows sample **debug x25 events** output.

Figure 2-132 Sample Debug X25 Events Output

```
router# debug x25 events

Serial2: X25 I R3 RESTART (5) 8 lci 0 cause 0 diag 0
Serial2: X25 I P1 CALL REQUEST (11) 8 lci 1024
From (2): 49 To(2): 46
  Facilities: (0)
  Call User Data (4): 0xCC 00 00 00 (ip)
Serial2: X25 O P4 CALL CONNECTED (3) 8 lci 1024
Serial2: X25 I D1 CLEAR REQUEST (5) 8 lci 1024 cause 0 diag 122
Serial2: X25 O D1 CLEAR CONFIRMATION (3) 8 lci 1024
Serial2: X25 O D1 RESET REQUEST (5) 8 lci 1 cause 0 diag 122
Serial2: X25 I D1 RESET CONFIRMATION (3) 8 lci 1
```

See the **debug x25 all** command description for information on the fields in **debug x25 events** output.

debug x25 vc

Use the **debug x25 vc** EXEC command to display information on traffic for a particular virtual circuit in order to solve any connectivity or performance problems it is exhibiting. The **no** form of this command removes the filter for a particular virtual circuit from the **debug x25 all** or **debug x25 events** output.

debug x25 vc *number*
no debug x25 vc *number*

Syntax Description

number	VC number associated with the virtual circuit(s) you want to monitor

Command Mode

EXEC

Usage Guidelines

Because no interface is specified, traffic on any VC that has the specified *number* is reported.

The **debug x25 vc** command limits the output of **debug x25 all** or **debug x25 events** output to the packets occurring on a particular VC number. This command modifies the operation of the **debug x25 all** or **debug x25 events** commands, so one of those commands must be used with **debug x25 vc** to produce output.

VC 0 cannot be specified. It is used for X.25 service messages, such as RESTART packets, not VC traffic. VC0 can be monitored only when no VC filter is used.

Note Because **debug x25 vc** only displays traffic for a small subset of virtual circuits, it is safe to use even under heavy traffic conditions, as long as events for that virtual circuit are fewer than 25 packets per second.

Sample Display

Figure 2-133 shows sample **debug x25 vc** output.

Figure 2-133 Sample Debug X25 VC Output

```
router# debug x25 vc 1
X25 debugging output restricted to VC1
router# debug x25 events
X25 special event debugging is on
router# show debug
X.25 (debugging restricted to VC number 1):
 X25 special event debugging is on

Serial0: X25 0 P2 CALL REQUEST (19) 8 lci 1
 From(14): 31250000000101 To(14): 31109090096101
  Facilities (0)
Serial0: X25 I P2 CLEAR REQUEST (5) 8 lci 1 cause diag 122
```

See the **debug x25 all** command description for information on the fields in **debug x25 vc** output.

debug xns packet

Use the **debug xns packet** EXEC command to display information on XNS packet traffic, including the addresses for source, destination, and next hop router of each packet. The **no** form of this command disables debugging output.

debug xns packet
no debug xns packet

Syntax Description

This command has no arguments or keywords.

Command Mode

EXEC

Usage Guidelines

To gain the fullest understanding of XNS routing activity, you should enable **debug xns routing** and **debug xns packet** together.

Sample Display

Figure 2-134 shows sample **debug xns packet** output.

Figure 2-134 Sample Debug XNS Packet Output.

```
router# debug xns packet

XNS: src=5.0000.0c02.6d04, dst=5.ffff.ffff.ffff, packet sent
XNS: src=1.0000.0c00.440f, dst=1.ffff.ffff.ffff, rcvd. on Ethernet0
XNS: src=1.0000.0c00.440f, dst=1.ffff.ffff.ffff, local processing
```

Table 2-89 describes significant fields shown in Figure 2-134.

Table 2-89 Debug XNS Packet Field Descriptions

Field	Description
XNS:	Indicates that this is an XNS packet.
src = 5.0000.0c02.6d04	Indicates that the source address for this message is 0000.0c02.6d04 on network 5.
dst = 5.ffff.ffff.ffff	Indicates that the destination address for this message is the broadcast address ffff.ffff.ffff on network 5.
packet sent	Indicates that the packet to destination address 5.ffff.ffff.ffff in Figure 2-134, as displayed using the **debug xns packet** command, was queued on the output interface.
rcvd. on Ethernet0	Indicates that the router just received this packet through the Ethernet0 interface.
local processing	Indicates that the router has examined the packet and determined that it must process it, rather than forwarding it.

debug xns routing

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

debug xns routing
no debug xns routing

Syntax Description

This command has no arguments or keywords.

Command Mode

EXEC

Usage Guidelines

To gain the fullest understanding of XNS routing activity, enable **debug xns routing** and **debug xns packet** together.

Sample Display

Figure 2-135 shows sample **debug xns routing** output.

Figure 2-135 Sample Debug XNS Routing Output

```
router# debug xns routing

XNSRIP: sending standard periodic update to 5.ffff.ffff.ffff via Ethernet2
  network 1, hop count 1
  network 2, hop count 2

XNSRIP: got standard update from 1.0000.0c00.440f socket 1 via Ethernet0
  net 2: 1 hops
```

Table 2-90 describes significant fields shown in Figure 2-135.

Table 2-90 Debug XNS Routing Field Descriptions

Field	Description
XNSRIP:	This is an XNS routing packet.
sending standard periodic update	Router indicates that this is a periodic XNS routing information update.
to 5.ffff.ffff.ffff	Destination address is ffff.ffff.ffff on network 5.
via Ethernet2	Name of the output interface.
network 1, hop count 1	Network 1 is one hop away from this router.
got standard update from 1.0000.0c00.440f	Router indicates that it has received an XNS routing information update from address 0000.0c00.440f on network 1.
socket 1	The socket number is a well-known port for XNS. Possible values include 1—routing information 2—echo 3—router error

APPENDIX A

X.25 Cause and Diagnostic Codes

This appendix covers the X.25 cause and diagnostic codes, as referred to in the **debug x25 all** command of the "Debug Commands" chapter. For more information on these codes, see the 1984 ITU-T X.25 Recommendation.

Note The ITU-T carries out the functions of the former Consultative Committee for International Telegraph and Telephone (CCITT).

Note The router reports the decimal value of a cause or diagnostic code, whereas other X.25 equipment may report these codes in hexadecimal notation. For this reason, this appendix lists both the decimal and hexadecimal values of the cause and diagnostic codes.

Table A-1 describes the differences between our implementation of certain X.25 network-generated, "international problem" diagnostic fields and the definitions provided in Annex E of ITU-T Recommendation X.25. The Annex E Table E-1/X.25 includes the complete diagnostic field listing.

Table A-1 Annex E International Problem Diagnostic Code Differences

Decimal Value	Annex E, Rec. X.25 Diagnostic Description	Cisco Proprietary Definition of Diagnostic Codes
112	International problem	Not used.
113	Remote network problem	Not used.
114	International protocol problem	Not used.
115	International link out of order	Indicates one of the following failures: failed when initializing a switched PVC; in TCP tunneling, failed when initiating or resetting a PVC; or, failed when PAD PVC circuit was initiated or reset.
116	International link busy	Not used.
117	Transit network facility problem	Not used.
118	Remote network facility problem	Not used.

Decimal Value	Annex E, Rec. X.25 Diagnostic Description	Cisco Proprietary Definition of Diagnostic Codes
119	International routing problem	Indicates the following failure: in TCP tunneling of X.25 when session is closed by network.
120	Temporary routing problem	Indicates the following failure: when tunneling X.25 through TCP/IP and the remote network is identified as unreachable.
121	Unknown called DNIC	Not used.
122	Maintenance action (may apply to maintenance action within a national network	For CMNS, indicates the following: router fails to route the call due to setup or unreachability of destination; when VC is cleared using the **clear x25-vc** EXEC command; when router CLEARs a VC when its idle timer expires.

X.25 Cause Codes

A cause code indicates an event that triggered an X.25 packet. The cause code can only appear in entries for CLEAR REQUEST, REGISTRATION CONFIRMATION, RESET REQUEST, and RESTART packets. Possible values for the cause code can vary, depending on the type of packet. Because the REGISTRATION exchange is not supported, those cause codes are not documented in this section.

Table A-2 describes the meanings of cause codes for CLEAR REQUEST packets.

Table A-2 Cause Code Descriptions for CLEAR REQUEST Packets

Code (Hex)	Code (Dec)	Description
00	0 (or 128 to 255)	DTE originated
01	1	Number busy
03	3	Invalid facility request
05	5	Network congestion
09	9	Out of order
0B	11	Access barred
0D	13	Not obtainable
11	17	Remote procedure error
13	19	Local procedure error
15	21	RPOA out of order
19	25	Reverse charging not accepted
21	33	Incompatible destination
29	41	Fast select not accepted
39	57	Ship absent

Table A-3 describes the meanings of cause codes for RESET REQUEST packets.

Table A-3 Cause Code Descriptions for RESET REQUEST Packets

Code (Hex)	Code (Dec)	Description
00	0 (or 128 to 255)	DTE originated
01	1	Out of order
03	3	Remote procedure error
05	5	Local procedure error
07	7	Network congestion
09	9	Remote DTE operational
0F	15	Network operational
11	17	Incompatible destination
1D	29	Network out of order

Table A-4 describes the meanings of cause codes for RESTART packets.

Table A-4 Cause Code Descriptions for RESTART Packets

Code (Hex)	Code (Dec)	Description
00	0 (or 128 to 255)	DTE restarting
01	1	Local procedure error
03	3	Network congestion
07	7	Network operational
7F	127	Registration/cancellation confirmed

X.25 Diagnostic Codes

The X.25 diag (diagnostic) code provides an additional hint as to what, if anything, went wrong. This code can only appear in entries for CLEAR REQUEST, DIAGNOSTIC, RESET REQUEST, and RESTART packets. Unlike the cause codes, the diag codes do not vary depending upon the type of packet.

Note These diagnostic codes can be produced by any equipment handling a given virtual circuit, and are then propagated through all equipment handling that virtual circuit. Thus, receipt of a diagnostic code may not indicate a problem with the router.

Table A-5 describes the meanings of possible diag codes.

Table A-5 X.25 Diagnostic Field Code Descriptions

Code (Hex)	Code (Dec)	Description
00	00	No additional information
01	01	Invalid P(S)
02	02	Invalid P(R)
10	16	Packet type invalid
11	17	Packet type invalid for state R1
12	18	Packet type invalid for state R2
13	19	Packet type invalid for state R3
14	20	Packet type invalid for state P1
15	21	Packet type invalid for state P2
16	22	Packet type invalid for state P3
17	23	Packet type invalid for state P4
18	24	Packet type invalid for state P5
19	25	Packet type invalid for state P6
1A	26	Packet type invalid for state P7

Code (Hex)	Code (Dec)	Description
1B	27	Packet type invalid for state D1
1C	28	Packet type invalid for state D2
1D	29	Packet type invalid for state D3
20	32	Packet not allowed
21	33	Unidentifiable packet
22	34	Call on one-way logical channel
23	35	Invalid packet type on a permanent virtual circuit
24	36	Packet on unassigned LCN
25	37	Reject not subscribed to
26	38	Packet too short
27	39	Packet too long
28	40	Invalid GFI (General Format Identifier)
29	41	Restart or registration packet with nonzero LCI
2A	42	Packet type not compatible with facility
2B	43	Unauthorized interrupt confirmation
2C	44	Unauthorized interrupt
2D	45	Unauthorized reject
30	48	Timer expired
31	49	Timer expired for incoming call
32	50	Timer expired for clear indication
33	51	Timer expired for reset indication
34	52	Timer expired for restart indication
35	53	Timer expired for call deflection
40	64	Call setup, clearing, or registration problem
41	65	Facility code not allowed
42	66	Facility parameter not allowed
43	67	Invalid called address
44	68	Invalid calling address
45	69	Invalid facility length
46	70	Incoming call barred
47	71	No logical channel available
48	72	Call collision
49	73	Duplicate facility requested
4A	74	Nonzero address length
4B	75	Nonzero facility length
4C	76	Facility not provided when expected
4D	77	Invalid ITU-T-specified DTE facility
4E	78	Maximum number of call redirections or deflections exceeded

Code (Hex)	Code (Dec)	Description
50	80	Miscellaneous
51	81	Improper cause code for DTE
52	82	Octet not aligned
53	83	Inconsistent Q bit setting
54	84	NUI (Network User Identification) problem
70	112	International problem
71	113	Remote network problem
72	114	International protocol problem
73	115	International link out of order
74	116	International link busy
75	117	Transit network facility problem
76	118	Remote network facility problem
77	119	International routing problem
78	120	Temporary routing problem
79	121	Unknown called DNIC
7A	122	Maintenance action (**clear x25 vc** command issued)

Diagnostic codes with values of 80 or greater in hexadecimal, or with values of 128 or greater in decimal, are specific to a particular network. To learn the meanings of these codes, contact the administrator for that network.

APPENDIX B

ISDN Switch Types, Codes, and Values

This appendix contains a list of the supported switch types. It also contains the ISDN cause codes, ISDN bearer capability values, and progress description field values that are valid within the debug commands for ISDN.

Note The ITU-T carries out the functions of the former Consultative Committee for International Telegraph and Telephone (CCITT).

Table B-1 lists the ISDN switch types supported by the ISDN interface.

Table B-1 Supported ISDN Switch Types

Identifier	Description
basic-1tr6	German 1TR6 ISDN switches
basic-5ess	AT&T basic rate switches
basic-dms100	NT DMS-100 basic rate switches
basic-net3	NET3 ISDN switches (UK and others)
basic-ni1	National ISDN-1 switches
basic-nwnet3	Norway Net3 switches
basic-nznet3	New Zealand Net3 switches
basic-ts013	Australian TS013 switches
none	No switch defined
ntt	Japanese NTT ISDN switches
primary-4ess	AT&T 4ESS switch type for the U.S. (ISDN PRI only)
primary-5ess	AT&T 5ESS switch type for the U.S. (ISDN PRI only)
primary-dms100	NT DMS-100 switch type for the U.S. (ISDN PRI only)
vn2	French VN2 ISDN switches
vn3	French VN3 ISDN switches
primary-ntt	INS-Net 1500 for Japan
primary-net5	NET5 ISDN PRI switches (Europe)

Table B-2 lists the ISDN cause code fields that display in the following format within the debug commands:

```
i=0xy1y2z1z2a1a2
```

Table B-2 ISDN Cause Code Fields

Field	Value—Description
0x	The values that follow are in hexadecimal.
y1	8—ITU-T standard coding.
y2	0—User 1—Private network serving local user 2—Public network serving local user 3—Transit network 4—Public network serving remote user 5—Private network serving remote user 7—International network A—Network beyond internetworking point
z1	Class of cause value.
z2	Value of cause value.
a1	(Optional) Diagnostic field that is always 8.
a2	(Optional) Diagnostic field that is one of the following values: 0—Unknown 1—Permanent 2—Transient

Table B-3 lists descriptions of the cause value field of the cause information element. The notes referred to in the Diagnostics column follow the table.

Table B-3 ISDN Cause Values

Cause Value		Cause		
Class	Value	Number	Cause	Diagnostics
0 0 0	0 0 0 1	1	Unallocated (unassigned) number	Note 12
0 0 0	0 0 1 0	2	No route to specified transit network	Transit network identity (Note 11)
0 0 0	0 0 1 1	3	No route to destination	Note 12
0 0 0	0 1 1 0	6	Channel unacceptable	
0 0 0	0 1 1 1	7	Call awarded and being delivered in an established channel	
0 0 1	0 0 0 0	16	Normal call clearing	Note 12
0 0 1	0 0 0 1	17	User busy	
0 0 1	0 0 1 0	18	No user responding	
0 0 1	0 0 1 1	19	No answer from user (user alerted)	

Cause Value		Cause		
Class	**Value**	**Number**	**Cause**	**Diagnostics**
0 0 1	0 1 0 1	21	Call rejected	Note 12. User supplied diagnostic (Note 4)
0 0 1	0 1 1 0	22	Number changed	
0 0 1	1 0 1 0	26	Non-selected user clearing	
0 0 1	1 0 1 1	27	Designation out of order	
0 0 1	1 1 0 0	28	Invalid number format	
0 0 1	1 1 0 1	29	Facility rejected	Facility identification (Note 1)
0 0 1	1 1 1 0	30	Response to STATUS ENQUIRY	
0 0 1	1 1 1 1	31	Normal, unspecified	
0 1 0	0 0 1 0	34	No circuit/channel available	Note 10
0 1 0	0 1 1 0	38	Network out of order	
0 1 0	1 0 0 1	41	Temporary failure	
0 1 0	1 0 1 0	42	Switching equipment congestion	
0 1 0	1 0 1 1	43	Access information discarded	Discarded information element identifier(s) (Note 6)
0 1 0	1 1 0 0	44	Requested circuit/channel not available	Note 10
0 1 0	1 1 1 1	47	Resources unavailable, unspecified	
0 1 1	0 0 0 1	49	Quality of service unavailable	Table B-2
0 1 1	0 0 1 0	50	Requested facility not subscribed	Facility identification (Note 1)
0 1 1	1 0 0 1	57	Bearer capability not authorized	Note 3
0 1 1	1 0 1 0	58	Bearer capability not presently available	Note 3
0 1 1	1 1 1 1	63	Service or option not available, unspecified	
1 0 0	0 0 0 1	65	Bearer capability not implemented	Note 3
1 0 0	0 0 1 0	66	Channel type not implemented	Channel Type (Note 7)
1 0 0	0 1 0 1	69	Requested facility not implemented	Facility Identification (Note 1)
1 0 0	0 1 1 0	70	Only restricted digital information bearer capability is available	
1 0 0	1 1 1 1	79	Service or option not implemented, unspecified	
1 0 1	0 0 0 1	81	Invalid call reference value	
1 0 1	0 0 1 0	82	Identified channel does not exist	Channel identity
1 0 1	0 0 1 1	83	A suspended call exists, but this call identity does not	
1 0 1	0 1 0 0	84	Call identity in use	
1 0 1	0 1 0 1	85	No call suspended	
1 0 1	0 1 1 0	86	Call having the requested call identity has been cleared	Clearing cause
1 0 1	1 0 0 0	88	Incompatible destination	Incompatible parameter (Note 2)

Cause Value		Cause		
Class	Value	Number	Cause	Diagnostics
1 0 1	1 0 1 1	91	Invalid transit network selection	
1 0 1	1 1 1 1	95	Invalid message, unspecified	
1 1 0	0 0 0 0	96	Mandatory information element is missing	Information element identifier(s) (Note 6)
1 1 0	0 0 0 1	97	Message type non-existent or not implemented	Message type
1 1 0	0 0 1 0	98	Message not compatible with call state or message type non-existent or not implemented	Message type
1 1 0	0 0 1 1	99	Information element non-existent or not implemented	Information element identifier(s) (Notes 6, 8)
1 1 0	0 1 0 0	100	Invalid information element contents	Information element identifier(s) (Note 6)
1 1 0	0 1 0 1	101	Message not compatible with call state	Message type
1 1 0	0 1 1 0	102	Recovery on timer expires	Timer number (Note 9)
1 1 0	1 1 1 1	111	Protocol error, unspecified	
1 1 1	1 1 1 1	127	Internetworking, unspecified	

Note 1: The coding of facility identification is network dependent.

Note 2: Incompatible parameter is composed of incompatible information element identifier.

Note 3: The format of the diagnostic field for cause 57, 58, and 65 is shown in the ITU-T Q.931 specification.

Note 4: User-supplied diagnostic field is encoded according to the user specification, subject to the maximum length of the cause information element. The coding of user-supplied diagnostics should be made in such a way that it does not conflict with the coding described in Table B-2.

Note 5: New destination is formatted as the called party number information element, including information element identifier. Transit network selection may also be included.

Note 6: Locking and non-locking shift procedures described in the ITU-T Q.931 specification apply. In principle, information element identifiers are in the same order as the information elements in the received message.

Note 7: The following coding is used:

- Bit 8—extension bit
- Bit 7 through 5—spare
- Bit 4 through 1—according to Table 4-15/Q.931 octet 3.2, channel type in ITU-T Q.931 specification

Note 8: When only locking shift information element is included and no variable length information element identifier follows, it means that the codeset in the locking shift itself is not implemented.

Note 9: The timer number is coded in IA5 characters. The following coding is used in each octet:

- Bit 8—Spare “0”
- Bit 7 through 1—IA5 character

Note 10: Examples of the cause values to be used for various busy/congestion condition appear in Annex J of the ITU-T Q.931 specification.

Note 11: The diagnostic field contains the entire transit network selection or network-specific facilities information element, as applicable.

Table B-4 lists the ISDN bearer capability values that display in the following format within the debug commands:

```
0x8890 for 64Kbps or 0x218F for 56 Kbps
```

Table B-4 ISDN Bearer Capability Values

Field	Value—Description
0x	Indication that the values that follow are in hexadecimal
88	ITU-T coding standard; unrestricted digital information
90	Circuit mode, 64 Kbps
21	Layer 1, V.110/X.30
8F	Synchronous, no in-band negotiation, 56Kbps

Table B-5 lists the values of the Progress description field contained in the ISDN Progress indicator information element.

Table B-5 Progress Description Field Values

Bits	Number	Description
0000001	1	Call is not end-to-end ISDN, further call progress information may be available in-band
0000010	2	Destination address is non-ISDN
0000011	3	Origination address is non-ISDN
0000100	4	Call has returned to the ISDN
0001000	8	In-band information or appropriate pattern now available.

All other values for the progress description field are reserved.

INDEX

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Z