

# Debug Commands

---

This chapter contains an alphabetical listing of the **debug** commands and their descriptions. 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 commands 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 **debug** command output is documented also varies. For example, the output for **debug** commands that generate lines of text is usually described line by line, and the output for **debug** commands that generate information in field format is usually described in tables.

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 aaa accounting

Use the **debug aaa accounting** EXEC command to display information on accountable events as they occur. Use the **no** form of the command to disable debugging output.

**[no] debug aaa accounting**

### Usage Guidelines

The information displayed by the **debug aaa accounting** command is independent of the accounting protocol used to transfer the accounting information to a server. Use the **debug tacacs** and **debug radius** protocol specific commands to get more detailed information about protocol-level issues.

You can also use the **show accounting** command to step through all active sessions and to print all the accounting records for actively accounted functions. The **show accounting** command allows you to display the active “accountable events” on the system. It provides systems administrators a quick look at what is going on, and may also be useful for collecting information in the event of a data loss of some kind on the accounting server. The **show accounting** command displays additional data on the internal state of the Authentication, Authorization, and Accounting (AAA) security system if **debug aaa accounting** is turned on as well.

### Sample Display

Figure 2-1 shows sample output from the **debug aaa accounting** command.

**Figure 2-1 Sample Debug AAA Accounting Output**

```
Router# debug aaa accounting

16:49:21: AAA/ACCT: EXEC acct start, line 10
16:49:32: AAA/ACCT: Connect start, line 10, glare
16:49:47: AAA/ACCT: Connection acct stop:
task_id=70 service=exec port=10 protocol=telnet address=172.31.3.78 cmd=glare
bytes_in=308 bytes_out=76 paks_in=45 paks_out=54 elapsed_time=14
```

### Related Commands

**debug aaa authentication**

**debug aaa authorization**

**debug radius**

**debug tacacs**

## debug aaa authentication

Use the **debug aaa authentication** EXEC command to display information on AAA/Terminal Access Controller Access Control System Plus (TACACS+) authentication. Use the **no** form of the command to disable debugging output.

**[no] debug aaa authentication**

### Usage Guidelines

Use this command to see what methods of authentication are being used and what the results of these methods are.

### Sample Display

Figure 2-2 shows sample **debug aaa authentication** output. A single EXEC login that uses the “default” method list and the first method, TACACS+, is displayed. The TACACS+ server sends a GETUSER request to prompt for the username and then a GETPASS request to prompt for the password, and finally a PASS response to indicate a successful login. The number 50996740 is the session ID, which is unique for each authentication. Use this ID number to distinguish between different authentications if several are occurring concurrently.

**Figure 2-2 Sample Debug AAA Authentication Output**

```
Router# debug aaa authentication

6:50:12: AAA/AUTHEN: create_user user='' ruser='' port='tty19' rem_addr='172.31.60.15'
authen_type=1 service=1 priv=1
6:50:12: AAA/AUTHEN/START (0): port='tty19' list='' action=LOGIN service=LOGIN
6:50:12: AAA/AUTHEN/START (0): using "default" list
6:50:12: AAA/AUTHEN/START (50996740): Method=TACACS+
6:50:12: TAC+ (50996740): received authen response status = GETUSER
6:50:12: AAA/AUTHEN (50996740): status = GETUSER
6:50:15: AAA/AUTHEN/CONT (50996740): continue_login
6:50:15: AAA/AUTHEN (50996740): status = GETUSER
6:50:15: AAA/AUTHEN (50996740): Method=TACACS+
6:50:15: TAC+: send AUTHEN/CONT packet
6:50:15: TAC+ (50996740): received authen response status = GETPASS
6:50:15: AAA/AUTHEN (50996740): status = GETPASS
6:50:20: AAA/AUTHEN/CONT (50996740): continue_login
6:50:20: AAA/AUTHEN (50996740): status = GETPASS
6:50:20: AAA/AUTHEN (50996740): Method=TACACS+
6:50:20: TAC+: send AUTHEN/CONT packet
6:50:20: TAC+ (50996740): received authen response status = PASS
6:50:20: AAA/AUTHEN (50996740): status = PASS
```

## debug aaa authorization

Use the **debug aaa authorization** EXEC command to display information on AAA/TACACS+ authorization. Use the **no** form of the command to disable debugging output.

**[no] debug aaa authorization**

### Usage Guidelines

Use this command to see what methods of authorization are being used and what the results of these methods are.

### Sample Display

Figure 2-3 shows sample **debug aaa authorization** output. In this display, an EXEC authorization for user “carrel” is performed. On the first line, the username is authorized. On the second and third lines, the AV (attribute value) pairs are authorized. The debug output displays a line for each attribute value pair that is authenticated. Next, the display indicates the authorization method used. The final line in the display indicates the status of the authorization process, in this case, a failure.

**Figure 2-3 Sample Debug AAA Authorization Output**

```
Router# debug aaa authorization

2:23:21: AAA/AUTHOR (0): user='carrel'
2:23:21: AAA/AUTHOR (0): send AV service=shell
2:23:21: AAA/AUTHOR (0): send AV cmd*
2:23:21: AAA/AUTHOR (342885561): Method=TACACS+
2:23:21: AAA/AUTHOR/TAC+ (342885561): user=carrel
2:23:21: AAA/AUTHOR/TAC+ (342885561): send AV service=shell
2:23:21: AAA/AUTHOR/TAC+ (342885561): send AV cmd*
2:23:21: AAA/AUTHOR (342885561): Post authorization status = FAIL
```

The **aaa authorization** command causes a request packet containing a series of attribute value pairs to be sent to the TACACS daemon as part of the authorization process. The daemon responds in one of the following three ways:

- Accepts the request as is
- Makes changes to the request
- Refuses the request, thereby refusing authorization

Table 2-1 describes attribute value pairs associated with the **aaa authorization** command that may show up in the debug output.

**Table 2-1 Attribute Value Pairs for Authorization**

Attribute Value	Description
service=arap	Authorization for AppleTalk Remote Access is being requested.
service=shell	Authorization for EXEC startup and command authorization is being requested.
service=ppp	Authorization for PPP is being requested.
service=slip	Authorization for SLIP is being requested.
protocol=lcp	Authorization for LCP is being requested (lower layer of PPP).
protocol=ip	Used with service=slip and service=slip to indicate which protocol layer is being authorized.
protocol=ipx	Used with service=ppp to indicate which protocol layer is being authorized.
protocol=atalk	Used with service=ppp or service=arap to indicate which protocol layer is being authorized.
protocol=vines	Used with service=ppp for VINES over PPP.
protocol=unknown	Used for undefined or unsupported conditions.
cmd=x	Used with service=shell, if cmd=NULL, this is an authorization request to start an EXEC. If cmd is not NULL, this is a command authorization request and will contain the name of the command being authorized. For example, cmd=telnet.
cmd-arg=x	Used with service=shell. When performing command authorization, the name of the command is given by a cmd=x pair for each argument listed. For example, cmd-arg=archie.sura.net.
acl=x	Used with service=shell and service=arap. For ARA, this pair contains an access list number. For service=shell, this pair contains an access class number. For example, acl=2.
inacl=x	Used with service=ppp and protocol=ip. Contains an IP input access list for SLIP or PPP/IP. For example, inacl=2.
outacl=x	Used with service=ppp and protocol=ip. Contains an IP output access list for SLIP or PPP/IP. For example, outacl=4.
addr=x	Used with service=slip, service=ppp, and protocol=ip. Contains the IP address that the remote host should use when connecting via SLIP or PPP/IP. For example, addr=172.30.23.11.
routing=x	Used with service=slip, service=ppp, and protocol=ip. Equivalent in function to the /routing flag in SLIP and PPP commands. Can either be true or false. For example, routing=true.
timeout=x	Used with service=arap. The number of minutes before an ARA session disconnects. For example, timeout=60.
autocmd=x	Used with service=shell and cmd=NULL. Specifies an autocommand to be executed at EXEC startup. For example, autocmd=telnet foo.com.
noescape=x	Used with service=shell and cmd=NULL. Specifies a noescape option to the username configuration command. Can be either true or false. For example, noescape=true.
nohangup=x	Used with service=shell and cmd=NULL. Specifies a nohangup option to the username configuration command. Can be either true or false. For example, nohangup=false.
priv-lvl=x	Used with service=shell and cmd=NULL. Specifies the current privilege level for command authorization as a number from 0 to 15. For example, priv-lvl=15.

**Table 2-1      Attribute Value Pairs for Authorization (Continued)**

<b>Attribute Value</b>	<b>Description</b>
zonelist= <i>x</i>	Used with service=arap. Specifies an AppleTalk zonelist for ARA. For example, zonelist=5.
addr-pool= <i>x</i>	Used with service=ppp and protocol=ip. Specifies the name of a local pool from which to get the address of the remote host.

## 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.

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

### Syntax Description

*type* (Optional) Interface type.

*number* (Optional) Interface number.

### 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 2-4 shows sample **debug apple arp** output.

**Figure 2-4 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 2-4 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 activities. The **no** form of this command disables debugging output.

**[no] debug apple domain**

### Usage Guidelines

Use the **debug apple domain** command to observe activity for 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, and updating a domain.

### Sample Display

Figure 2-5 shows sample **debug apple domain** output intermixed with output from the **debug apple remap** command; the two commands show related events.

**Figure 2-5 Sample Debug Apple Domain Output**

```
Router# debug apple domain
Router# debug apple remap

AT-REMAP: RemapProcess for net 30000 domain AURP Domain 1
AT-REMAP: ReshuffleRemapList for subdomain 1
AT-REMAP: Could not find a remap for cable 3000-3001
AT-DOMAIN: atdomain_DisablePort for Tunnel0
AT-DOMAIN: CleanupDomain for domain 1 [AURP Domain 1]
AT-DOMAIN: Disabling interface Ethernet1
AT-DOMAIN: atdomain_DisablePort for Ethernet1
AT-DOMAIN: CleanupDomain for domain 1 [AURP 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: RemapProcess for net 30000 domain AURP Domain 1 Remaped Net 10000
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 [AURP 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
```

### Related Command

**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.

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

### Syntax Description

*type* (Optional) Interface type.  
*number* (Optional) Interface number.

### 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 2-7 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 2-7 Sample Debug Apple Errors Output**

```
Router# debug apple errors
```

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

As Figure 2-7 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.

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

### Syntax Description

<i>type</i>	(Optional) Interface type.
<i>number</i>	(Optional) Interface number.

### 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 Displays

Figure 2-8 shows sample **debug apple events** output that describes a nonseed router coming up in discovery mode.

**Figure 2-8 Sample Debug Apple Events Output—Discovery Mode State Changes**

```

router# debug apple events
Discovery mode state changes
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-6-ACQUIREMODE: Ether0: AT port initializing; acquiring net configuration
Ether0: AppleTalk state changed; acquiring -> restarting
Ether0: AppleTalk state changed; restarting -> line down
Ether0: AppleTalk state changed; line down -> restarting
Ether0: AppleTalk state changed; restarting -> probing
%AT-6-ADDRUSED: Ether0: AppleTalk node up; using address 4160.148
Ether0: AppleTalk state changed; probing -> acquiring
%AT-6-ACQUIREMODE: Ether0: AT port initializing; acquiring net configuration
Ether0: AppleTalk state changed; acquiring -> requesting zones
Ether0: AT: Resetting interface address filters
%AT-5-INTRESTART: Ether0: AppleTalk port restarting; protocol restarted
Ether0: AppleTalk state changed; requesting zones -> verifying
AT: Sent GetNetInfo request broadcast on Ethernet0
Ether0: AppleTalk state changed; verifying -> checking zones
Ether0: AppleTalk state changed; checking zones -> operational

```

S25M2

As Figure 2-8 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 2-8 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 2-9 shows sample **debug apple events** output that describes a nondiscovery-enabled router coming up when no other router is on the wire.

**Figure 2-9 Sample Debug Apple Events Output—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

```

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

S2543

As Figure 2-9 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 2-9 indicates this situation.

Figure 2-10 shows sample **debug apple events** output that describes a discovery-enabled router coming up when there is no seed router on the wire.

**Figure 2-10 Sample Debug Apple Events Output—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 2-10 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 2-11 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 2-11 Sample Debug Apple Events Output—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

```

Indicates configuration mismatch

S2545

The following three configuration command lines indicate the part of the router's configuration that caused the configuration mismatch shown in Figure 2-11:

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

The router shown in Figure 2-11 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 "Marketing."

## 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.

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

### Syntax Description

*type* (Optional) Interface type.

*number* (Optional) Interface number.

### 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 2-12 shows sample **debug apple nbp** output.

**Figure 2-12 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 2-12 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 2-3 describes the fields in the first line of output shown in Figure 2-12.

**Table 2-3 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 2-4 describes the fields in the second line of output shown in Figure 2-12.

**Table 2-4 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 2-12 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 2-12 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.

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

### Syntax Description

*type* (Optional) Interface type.  
*number* (Optional) Interface number.

### 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 2-13 shows sample **debug apple packet** output.

**Figure 2-13 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 2-5 describes the fields in the first line of output shown in Figure 2-13.

**Table 2-5 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
encype SNAP	Encapsulation type for the packet
size 60	Size of the packet (in bytes)
encaps000000000000000000000000	Encapsulation

Table 2-6 describes the fields in the second line of output shown in Figure 2-13.

**Table 2-6 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 2-13 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 activities. The **no** form of this command disables debugging output.

**[no] debug apple remap**

### 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 2-14 shows sample **debug apple remap** output intermixed with output from the **debug apple domain** command; the two commands show related events.

**Figure 2-14 Sample Debug Apple Remap and Domain Output**

```
Router# debug apple remap
Router# debug apple domain

AT-REMAP: RemapProcess for net 30000 domain AURP Domain 1
AT-REMAP: ReshuffleRemapList for subdomain 1
AT-REMAP: Could not find a remap for cable 3000-3001
AT-DOMAIN: atdomain_DisablePort for Tunnel0
AT-DOMAIN: CleanUpDomain for domain 1 [AURP Domain 1]
AT-DOMAIN: Disabling interface Ethernet1
AT-DOMAIN: atdomain_DisablePort for Ethernet1
AT-DOMAIN: CleanUpDomain for domain 1 [AURP 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: RemapProcess for net 30000 domain AURP Domain 1 Remaped Net 10000
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 [AURP 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
```

### 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.

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

### Syntax Description

*type* (Optional) Interface type.

*number* (Optional) Interface number.

### 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 router CPU utilization is less than 50 percent.

---

### Sample Display

Figure 2-15 shows sample **debug apple routing** output.

**Figure 2-15 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 2-15 follow.

Table 2-7 describes the fields in the first line of sample **debug apple routing** output.

**Table 2-7 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

**Table 2-7      Debug Apple Routing Field Descriptions—Part 1 (Continued)**

Field	Description
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 2-8 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 2-8      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.

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

### Syntax Description

*type* (Optional) Interface type.

*number* (Optional) Interface number.

### 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 the **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 2-16 shows sample **debug apple zip** output.

**Figure 2-16 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 2-16 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 appn all

Use the **debug appn all** EXEC command to turn on all possible debugging messages for Advanced Peer-to-Peer Networking (APPN). The **no** form of this command disables debugging output.

**[no] debug appn all**

---

**Note** Refer to the other forms of the **debug appn** command to enable specific debug output selectively.

---

### Usage Guidelines

This command shows all APPN events. Use other forms of the **debug appn** command to display specific types of events.

---

**Note** Because the **debug appn all** command can generate many messages and alter timing in the network node, use it only when instructed by authorized support personnel.

---



**Caution** Debugging output takes priority over other network traffic. The **debug appn all** command generates more output than any other **debug appn** command and can alter timing in the network node. This command can severely diminish router performance or even render it unusable. In virtually all cases, it is best to use specific **debug appn** commands.

Refer to the documentation for specific **debug appn** commands for sample displays and explanations.

### Related Commands

**debug appn cs**  
**debug appn ds**  
**debug appn hpr**  
**debug appn ms**  
**debug appn nof**  
**debug appn pc**  
**debug appn ps**  
**debug appn scm**  
**debug appn ss**  
**debug appn trs**

## debug appn cs

Use the **debug appn cs** EXEC command to display APPN Configuration Services (CS) component activity. The **no** form of this command disables debugging output.

**[no] debug appn cs**

### Usage Guidelines

The Configuration Services (CS) component is responsible for defining link stations, ports, and connection networks. It is responsible for the activation and deactivation of ports and link stations and handles status queries for these resources.

### Sample Display

Figure 2-17 shows sample **debug appn cs** output. In this example a link station is being stopped.

**Figure 2-17 Sample Debug APPN CS Output**

```
Router# debug appn cs

Turned on event 008000FF

Router# appn stop link PATTY

APPN: ----- CS ----- Deq STOP_LS message
APPN: ----- CS ----- FSM LS: 75 17 5 8
APPN: ----- CS ----- Sending DEACTIVATE_AS - station PATTY
APPN: ----- CS ----- deactivate_as_p->ips_header.lpid = A80A60
APPN: ----- CS ----- deactivate_as_p->ips_header.lpid = A80A60
APPN: ----- CS ----- Sending DESTROY_TG to PC - station PATTY - lpid=A80A60
APPN: ----- CS ----- Deq DESTROY_TG - station PATTY
APPN: ----- CS ----- FSM LS: 22 27 8 0
APPN: ----- CS ----- Sending TG update for LS PATTY to TRS
APPN: ----- CS ----- ENTERING XID_PROCESSING: 4
%APPN-6-APPNSENDMSG: Link Station PATTY stopped
```

Table 2-9 shows describes the fields and messages shown in Figure 2-17.

**Table 2-9 Debug APPN CS Field Descriptions**

Field	Description
APPN	APPN debugging output.
CS	Configuration Services component output.
Deq	CS received a message from another component.
FSM LS	The link station finite state machine is being referenced.
Sending	CS is sending a message to another component.

### Related Command

**debug appn all**

## debug appn ds

Use the **debug appn ds** EXEC command to display debugging information on APPN Directory Services (DS) component activity. The **no** form of this command disables debugging output.

**[no] debug appn ds**

### Usage Guidelines

The Directory Services (DS) component manages searches for resources in the APPN network. DS is also responsible for registration of resources within the network.

### Sample Display

Figure 2-18 shows sample **debug appn ds** output. In this example a search has been received.

**Figure 2-18 Sample Debug APPN DS Output**

```
Router# debug appn ds
Turned on event 080000FF
APPN: NEWDS: LS: search from: NETA.PATTY
APPN: NEWDS: pcid: DD3321E8B5667111
APPN: NEWDS: Invoking FSM NNSolu
APPN: NEWDS: LSfsm_NNSolu: 00A67AA0 pcid: DD3321E8B5667111 row: 0 col: 0 inp: 80200000
APPN: NEWDS: LSfsm_parent: 00A89940 row: 0 col: 0 inp: 80000000
APPN: NEWDS: Rcvd a LMRQ
APPN: NEWDS: LSfsm_NNSolu: 00A67AA0 pcid: DD3321E8B5667111 row: 12 col: 1 inp: 40000000
APPN: NEWDS: LSfsm_parent: 00A89940 row: 8 col: 1 inp: 40000000
APPN: NEWDS: LSfsm_child: 00A89BE8 row: 0 col: 0 inp: 80000080
APPN: NEWDS: PQenq REQUEST_ROUTE(RQ) to TRS
APPN: NEWDS: LSfsm_child: 00A8A1C0 row: 1 col: 0 inp: 80000008
APPN: NEWDS: LSfsm_NNSolu: 00A67AA0 pcid: DD3321E8B5667111 row: 5 col: 1 inp: 80C04000
APPN: NEWDS: LSfsm_child: 00A8A1C0 row: 7 col: 1 inp: 80844008
APPN: NEWDS: Rcvd a LMRQ
APPN: NEWDS: LSfsm_NNSolu: 00A67AA0 pcid: DD3321E8B5667111 row: 16 col: 6 inp: 40800000
APPN: NEWDS: LSfsm_child: 00A8A1C0 row: 14 col: 5 inp: 40800000
APPN: NEWDS: LSfsm_parent: 00A89940 row: 3 col: 1 inp: 80840000
APPN: NEWDS: send locate to node: NETA.PATTY
```

Table 2-10 provides explanations for fields in the **debug appn ds** output shown in Figure 2-18.

**Table 2-10 Debug APPN DS Field Descriptions**

Field	Description
APPN	APPN debugging output.
NEWDS	Directory Services component output.
search from	A locate was received from NETA.PATTY.
LSfsm_	The Locate Search finite state machine is being referenced.
PQenq	A message was sent to another component.
Rcvd	A message was received from another component.
send locate	A locate will be sent to NETA.PATTY.

### Related Command

**debug appn all**

## debug appn hpr

Use the **debug appn hpr** privileged EXEC command to display debugging information related to High Performance Routing (HPR) code execution. The **no** form of this command disables debugging output.

**[no] debug appn hpr**

### Sample Display

Figure 2-19 shows sample output from the **debug appn hpr** command.

**Figure 2-19 Sample Debug APPN HPR Output**

```
Router# debug appn hpr

APPN: -- ncl.ncl_map_dlc_type() -- mapping TOKEN_RING(4) to NCL_TR(3)
APPN: -- ncl.ncl_port() -- called with port_type:3, cisco_idb:893A14, hpr_ssap:C8
APPN: -- ncl.process_port_change() -- port coming up
APPN: -- ncl.process_port_change() -- PORT_UP
APPN: -- ncl.ncl_port_fsm -- FSM Invoked: Input:0, State:0->1, Action:0
APPN: -- ncl.ncl_port_fsm -- FSM Invoked: Input:1, State:1->2, Action:1
APPN: -- ncl.ncl_unmap_dlc_type() -- mapping NCL(3) to CLS(3)
APPN: ----- ANR ----- Sending ACTIVATE_SAP.req
APPN: -- cswncsnd.main() -- received LSA_IPS ips.
APPN: -- ncl.ncl_port_fsm -- FSM Invoked: Input:3, State:2->3, Action:4
APPN: -- ncl.ncl_assign_anr() -- Assigned ANR,anr:8002
APPN: -- ncl.ncl_map_dlc_type() -- mapping TOKEN_RING(4) to NCL_TR(3)
APPN: -- ncl.ncl_populate_anr() -- anr:8002, dlc_type:3, idb 893A14
APPN: -- ncl.ncl_populate_anr() -- send anr_tbl_update to owning cswncsnd
APPN: -- ncl.ncl_ls_fsm -- FSM Invoked: Input:0, State:0->1, Action:0
APPN: ncl.ncl_send_reqopn_stn_req
APPN: -- ncl.ncl_unmap_dlc_type() -- mapping NCL(3) to CLS(3)
APPN: -- ncl.ncl_ls_fsm() -- send anr_tbl_update to owning cswncsnd
APPN: -- cswncsnd.main() -- received ANR_TBL_UPDATE ips.
APPN: -- cswncsnd.apply_anr_table_update() -- ANR:8002
APPN: -- cswncsnd.main() -- received ANR_TBL_UPDATE ips.
APPN: -- cswncsnd.apply_anr_table_update() -- ANR:8002
APPN: -- cswncsnd.main() -- received LSA_IPS ips.
APPN: -- ncl.ncl_ls_fsm -- FSM Invoked: Input:1, State:1->2, Action:1
APPN: -- ncl.ncl_ls_fsm -- P_CEP_ID:AAF638
APPN: -- ncl.ncl_ls_fsm() -- send anr_tbl_update to owning cswncsnd
APPN: -- cswncsnd.main() -- received ANR_TBL_UPDATE ips.
APPN: -- cswncsnd.apply_anr_table_update() -- ANR:8002
APPN: rtpm: rtp_send() sent data over connection B9D5E8
APPN: hpr timer: rtt start time clocked at 135952 ms
APPN: -- cswncsnd.main() -- received NCL_SND_MSG ips.
APPN: -- cswncsnd.process_nlp_from_rtp() -- label: 8002, send to p_cep 00AAF638.
APPN: hpr timer: rtt end time clocked at 135972 ms
APPN: hpr timer: round trip time measured at 20 ms
```

Table 2-11 describes the **debug appn hpr** fields.

**Table 2-11 Debug APPN HPR Field Descriptions**

Field	Description
APPN	APPN debugging output.
NCL	Network control layer debugging output. Network control layer is the component that deals with ANR packets.

**Table 2-11      Debug APPN HPR Field Descriptions (Continued)**

<b>Field</b>	<b>Description</b>
ncl_port_fsm	Network control layer port finite state machine has been invoked.
ncl_assign_anr	ANR label has been assigned to a activating link station.
ncl_populate_anr	System is updating the ANR record with information specific to the link station.
ncl_ls_fsm	Network control layer link finite state machine has been invoked.
rtp_send	RTP is about to send a packet.
hpr timer	Debugging output related to an HPR timer.
rtt start time	RTP is measuring the round-rip time for an HPR status request packet. This is the start time.
NCL_SND_MSG	Network control layer has been requested to send a packet.
process_nlp_from_rtp	Network control layer has been requested by RTP to send a packet.
rtt end time	RTP is measuring the round trip time for an HPR status request packet. This is the time.
round trip time	Round-trip time for this HPR status exchange has been computed.

**Related Command****debug appn all**

## debug appn ms

Use the **debug appn ms** EXEC command to display debugging information on APPN Management Services (MS) component activity. The **no** form of this command disables debugging output.

**[no] debug appn ms**

### Usage Guidelines

The Management Services (MS) component is responsible for generating, sending, and forwarding network management information in the form of traps and alerts to a network management focal point, such as Netview, in the APPN network.

### Sample Display

Figure 2-20 shows sample **debug appn ms** output. In this example an error occurred that caused an alert to be generated.

**Figure 2-20 Sample Debug APPN MS Output**

```
Router# debug appn ms

APPN: ----- MSS00 ---- Deq ALERT_MSU msg
APPN: --- MSP70 --- ALERT MV FROM APPN WITH VALID LGTH
APPN: --- MSCPL --- Find Active FP
APPN: --- MSP30 --- Entering Build MS Transport
APPN: --- MSP31 --- Entering Building Routing Info.
APPN: --- MSP34 --- Entering Build GDS
APPN: --- MSP32 --- Entering Building UOW correlator
APPN: --- MSP34 --- Entering Build GDS
APPN: --- MSP30 --- Building GDS 0x1310
APPN: --- MSP30 --- Building MS Transport
APPN: --- MSP72 --- ACTIVE FP NOT FOUND, SAVE ONLY
APPN: --- MSUTL --- UOW <= 60, ALL COPIED in extract_uow
APPN: --- MSCAT --- by enq_cached_ms QUEUE SIZE OF QUEUE after enq 4
```

Table 2-12 describes fields in the **debug appn ms** output shown in Figure 2-20.

**Table 2-12 Debug APPN MS Output Field Descriptions**

Field	Description
APPN	Indicates that this is APPN debugging output.
MSP	Indicates that this is MS component output.

### Related Command

**debug appn all**

## debug appn nof

Use the **debug appn nof** EXEC command to display debugging information on APPN Node Operator Facility (NOF) component activity. The **no** form of this command disables debugging output.

**[no] debug appn nof**

### Usage Guidelines

The Node Operator Facility (NOF) component is responsible for processing commands entered by the user such as start, stop, show, and configuration commands. NOF forwards these commands to the proper component and wait for the response.

### Sample Display

Figure 2-21 shows sample **debug appn nof** output. In this example, an APPN connection network is being defined.

**Figure 2-21 Sample Debug APPN NOF Output**

```
Router# debug appn nof
Turned on event 010000FF

Router# config term

Enter configuration commands, one per line. End with CNTL/Z.
Router(config)# appn connection-network NETA.CISCO
Router(config-appn-cn)# port TR0
Router(config-appn-cn)# complete
router(config)#

APPN: ----- NOF ----- Define Connection Network Verb Received
APPN: ----- NOF ----- send define_cn_t ips to cs
APPN: ----- NOF ----- waiting for define_cn rsp from cs
router(config)#
```

Table 2-13 describes fields in the **debug appn nof** output shown in Figure 2-21.

**Table 2-13 Debug APPN NOF Field Descriptions**

Field	Description
APPN	APPN debugging output.
NOF	NOF component output.
Received	A configuration command was entered.
send	A message was sent to CS.
waiting	A response was expected from CS.

### Related Command

**debug appn all**

## debug appn pc

Use the **debug appn pc** EXEC command to display debugging information on APPN Path Control (PC) component activity. The **no** form of this command disables debugging output.

**[no] debug appn pc**

### Usage Guidelines

The Path Control (PC) component is responsible for passing Message Units (MUs) between the Data Link Control (DLC) layer and other APPN components. PC implements transmission priority by passing higher priority MUs to the DLC before lower priority MUs.

### Sample Display

Figure 2-22 shows sample **debug appn pc** output. In this example a MU is received from the network.

**Figure 2-22 Sample Debug APPN PC Output**

```
Router# debug appn pc

Turned on event 040000FF
APPN: ----- PC-----PC Deq REMOTE msg variant_name 2251
APPN: --PC-- mu received to PC lpid: A80AEC
APPN: --PC-- mu received from p_cep_id: 67C6F8
APPN: ----- PC-----PC Deq LSA_IPS from DLC
APPN: --PCX dequeued a DATA.IND
APPN: --- PC processing DL_DATA.ind
APPN: --PC-- mu_error_checker with no error, calling frr
APPN: --PC-- calling frr for packet received on LFSID: 1 2 3
APPN: ----- PC-----PC is sending MU to SC A90396
APPN: ----- SC-----send mu: A90396, rpc: 0, nws: 7, rh.b1: 90
APPN: SC: Send mu.snf: 8, th.b0: 2E, rh.b1: 90, dcf: 8
```

Table 2-14 describes fields in the **debug appn pc** output shown in Figure 2-22.

**Table 2-14 Debug APPN PC Field Descriptions**

Field	Description
APPN	APPN debugging output.
PC	PC component output.
Deq REMOTE	A message was received from the network.
mu received	The message is a MU.
DATA.IND	The MU contains data.
sending MU	The MU is session traffic for an ISR session. The MU is forwarded to the Session Connector component for routing.

### Related Command

**debug appn all**

## debug appn ps

Use the **debug appn ps** EXEC command to display debugging information on APPN Presentation Services (PS) component activity. The **no** form of this command disables debugging output.

**[no] debug appn ps**

### Usage Guidelines

The Presentation Services (PS) component is responsible for managing the Transaction Programs (TPs) used by APPN. TPs are used for sending and receiving searches, receiving resource registration, and sending and receiving topology updates.

### Sample Display

Figure 2-23 shows sample **debug appn ps** output. In this example a CP capabilities exchange is in progress.

**Figure 2-23 Sample Debug APPN PS Output**

```
Router# debug appn ps

Turned on event 200000FF
APPN: ---- CCA --- CP_CAPABILITIES_TP has started
APPN: ---- CCA --- About to wait for Partner to send CP_CAP
APPN: ---- CCA --- Partner LU name: NETA.PATTY
APPN: ---- CCA --- Mode Name: CPSVCMG
APPN: ---- CCA --- CGID: 78
APPN: ---- CCA --- About to send cp_cp_session_act to SS
APPN: ---- CCA --- Waiting for cp_cp_session_act_rsp from SS
APPN: ---- CCA --- Received cp_cp_session_act_rsp from SS
APPN: ---- CCA --- About to send CP_CAP to partner
APPN: ---- CCA --- Send to partner completed with rc=0, 0
APPN: ---- RCA --- Allocating conversation
APPN: ---- RCA --- Sending CP_CAPABILITIES
APPN: ---- RCA --- Getting conversation attributes
APPN: ---- RCA --- Waiting for partner to send CP_CAPABILITIES
APPN: ---- RCA --- Normal processing complete with cgid = 82
APPN: ---- RCA --- Deallocating CP_Capabilities conversation
```

Table 2-15 describes fields in the **debug appn ps** output shown in Figure 2-23.

**Table 2-15 Debug APPN PS Field Descriptions**

Field	Description
APPN	APPN debugging output.
CCA	CP Capabilities TP output.
RCA	Receive CP Capabilities TP output.

### Related Command

**debug appn all**

## debug appn scm

Use the **debug appn scm** EXEC command to display debugging information on APPN Session Connector Manager (SCM) component activity. The **no** form of this command disables debugging output.

**[no] debug appn scm**

### Usage Guidelines

The Session Connector Manager (SCM) component is responsible for the activation and deactivation the local resources that route an intermediate session through the router.

### Sample Display

Figure 2-24 shows sample **debug appn scm** output. In this example an intermediate session traffic is being routed.

**Figure 2-24 Sample Debug APPN SCM Output**

```
Router# debug appn scm

Turned on event 020000FF
Router#
APPN: ----- SCM-----SCM Deq a MU
APPN: ----- SCM-----SCM send ISR_INIT to SSI
APPN: ----- SCM----- (i05) Enter compare_fqpcid()
APPN: ----- SCM-----Adding new session_info table entry. addr=A93160
APPN: ----- SCM-----SCM Deq ISR_CINIT message
APPN: ----- SCM----- (i05) Enter compare_fqpcid()
APPN: ----- SCM-----SCM sends ASSIGN_LFSID to ASM
APPN: ----- SCM-----SCM Rcvd sync ASSIGN_LFSID from ASM
APPN: ----- SCM-----SCM PQenq a MU to ASM
APPN: ----- SCM-----SCM Deq a MU
APPN: ----- SCM----- (i05) Enter compare_fqpcid()
APPN: ----- SCM-----SCM PQenq BIND rsp to ASM
```

Table 2-16 describes fields in the **debug appn ps** output shown in Figure 2-24.

**Table 2-16 Debug APPN SCM Field Descriptions**

Field	Description
APPN	APPN debugging output.
SCM	SCM component output.

### Related Command

**debug appn all**

## debug appn ss

Use the **debug appn ss** EXEC command to display session services (SS) events. The **no** form of this command disables debugging output.

**[no] debug appn ss**

### Usage Guidelines

The Session Services (SS) component generates unique session identifiers, activates and deactivates control point-to-control point (CP-CP) sessions, and assists LUs in initiating and activating LU-LU sessions.

### Sample Display

Figure 2-25 shows sample **debug appn ss** output. In this example CP-CP sessions between the router and another node are being activated.

**Figure 2-25 Sample Debug APPN SS Output**

```
Router# debug appn ss

Turned on event 100000FF
APPN: ----- SS ----- Deq ADJACENT_CP_CONTACTED message
APPN: ----- SS ----- Deq SESSST_SIGNAL message
APPN: ----- SS ----- Deq CP_CP_SESSION_ACT message
APPN: Sending ADJACENT_NN_1015 to SCM, adj_node_p=A6B980,cp_name=NETA.PATTY
APPN: ----- SS ----- Sending REQUEST_LAST_FRSN message to TRS
APPN: ----- SS ----- Receiving REQUEST_LAST_FRSN_RSP from TRS
APPN: ----- SS ----- Sending ACTIVE CP_STATUS CONLOSER message to DS
APPN: ----- SS ----- Sending ACTIVE CP_STATUS CONLOSER message to MS
APPN: ----- SS ----- Sending ACTIVE CP_STATUS CONLOSER message to TRS
APPN: ----- SS ----- Sending CP_CP_SESSION_ACT_RSP message to CCA TP
APPN: ----- SS ----- Sending PENDING_ACTIVE CP_STATUS CONWINNER message to DS
APPN: ----- SS ----- Sending REQUEST_LAST_FRSN message to TRS
APPN: ----- SS ----- Receiving REQUEST_LAST_FRSN_RSP from TRS
APPN: ----- SS ----- Sending ACT_CP_CP_SESSION message to RCA TP
APPN: ----- SS ----- Deq ASSIGN_PCID message
APPN: ----- SS ----- Sending ASSIGN_PCID_RSP message to someone
APPN: ----- SS ----- Deq INIT_SIGNAL message
APPN: ----- SS ----- Sending REQUEST_COS_TPF_VECTOR message to TRS
APPN: ----- SS ----- Receiving an REQUEST_COS_TPF_VECTOR_RSP from TRS
APPN: ----- SS ----- Sending REQUEST_SINGLE_HOP_ROUTE message to TRS
APPN: ----- SS ----- Receiving an REQUEST_SINGLE_HOP_ROUTE_RSP from TRS
APPN: ----- SS ----- Sending ACTIVATE_ROUTE message to CS
APPN: ----- SS ----- Deq ACTIVATE_ROUTE_RSP message
APPN: ----- SS ----- Sending CINIT_SIGNAL message to SM
APPN: ----- SS ----- Deq ACT_CP_CP_SESSION_RSP message
APPN: -- SS----SS ssp00, act_cp_cp_session_rsp received, sense_code=0, cgid=5C,
ips@=A93790
APPN: Sending ADJACENT_NN_1015 to SCM, adj_node_p=A6B980,cp_name=18s
APPN: ----- SS ----- Sending ACTIVE CP_STATUS CONWINNER message to DS
APPN: ----- SS ----- Sending ACTIVE CP_STATUS CONWINNER message to MS
APPN: ----- SS ----- Sending ACTIVE CP_STATUS CONWINNER message to TRS
```

Table 2-17 describes fields in the **debug appn ss** output shown in Figure 2-25.

**Table 2-17      Debug APPN SS Field Descriptions**

<b>Field</b>	<b>Description</b>
APPN	APPN debugging output.
SS	SS component output.

**Related Command**

**debug appn all**

## debug appn trs

Use the **debug appn trs** EXEC command to display debugging information on APPN Topology and Routing Services (TRS) component activity. The **no** form of this command disables debugging output.

**[no] debug appn trs**

### Usage Guidelines

The Topology and Routing Services (TRS) component is responsible for creating and maintaining the topology database, creating and maintaining the class of service database, and computing and caching optimal routes through the network.

### Sample Display

Figure 2-26 shows sample **debug appn trs** output.

**Figure 2-26 Sample Debug APPN TRS Output**

```
Router# debug appn trs

Turned on event 400000FF
APPN: ----- TRS ----- Received a QUERY_CPNAME
APPN: ----- TRS ----- Received a REQUEST_ROUTE
APPN: ----- TRS ----- check_node node_name=NETA.LISA
APPN: ----- TRS ----- check_node node_index=0
APPN: ----- TRS ----- check_node node_weight=60
APPN: ----- TRS ----- add index 484 to origin description list
APPN: ----- TRS ----- add index 0 to dest description list
APPN: ----- TRS ----- origin tg_vector is NULL
APPN: ----- TRS ----- weight_to_origin = 0
APPN: ----- TRS ----- weight_to_dest = 0
APPN: ----- TRS ----- u_b_s_f weight = 30
APPN: ----- TRS ----- u_b_s_f prev_weight = 2147483647
APPN: ----- TRS ----- u_b_s_f origin_index = 484
APPN: ----- TRS ----- u_b_s_f dest_index = 0
APPN: ----- TRS ----- b_r_s_f weight = 30
APPN: ----- TRS ----- b_r_s_f origin_index = 484
APPN: ----- TRS ----- b_r_s_f dest_index = 0
APPN: ----- TRS ----- Received a REQUEST_ROUTE
APPN: ----- TRS ----- check_node node_name=NETA.LISA
APPN: ----- TRS ----- check_node node_index=0
APPN: ----- TRS ----- check_node node_weight=60
APPN: ----- TRS ----- check_node node_name=NETA.BART
APPN: ----- TRS ----- check_node node_index=484
APPN: ----- TRS ----- check_node node_weight=60
APPN: ----- TRS ----- add index 484 to origin description list
APPN: ----- TRS ----- add index 0 to dest description list
APPN: ----- TRS ----- origin_tg_weight to non-VN=30
APPN: ----- TRS ----- origin_node_weight to non-VN=60
APPN: ----- TRS ----- weight_to_origin = 90
APPN: ----- TRS ----- weight_to_dest = 0
APPN: ----- TRS ----- u_b_s_f weight = 120
APPN: ----- TRS ----- u_b_s_f prev_weight = 2147483647
APPN: ----- TRS ----- u_b_s_f origin_index = 484
APPN: ----- TRS ----- u_b_s_f dest_index = 0
APPN: ----- TRS ----- b_r_s_f weight = 120
APPN: ----- TRS ----- b_r_s_f origin_index = 484
APPN: ----- TRS ----- b_r_s_f dest_index = 0
```

Table 2-18 describes fields in the **debug appn trs** output shown in Figure 2-26.

**Table 2-18      Debug APPN TRS Field Descriptions**

<b>Field</b>	<b>Description</b>
APPN	APPN debugging output.
TRS	TRS component output.

Related Command

**debug appn all**

## debug arap

Use the **debug arap** EXEC command to display AppleTalk Remote Access Protocol (ARAP) events. The **no** form of this command disables debugging output.

```
[no] debug arap {internal | memory | mnp4 | v42bis} [[aux | console | vty ] linenum]
```

### Syntax Description

<b>internal</b>	Debug internal ARA packets.
<b>memory</b>	Debug memory allocation for ARA.
<b>mnp4</b>	Debug low-level asynchronous serial protocol.
<b>v42bis</b>	Debug V.42bis compression.
<b>aux</b>	(Optional) Auxiliary line.
<b>console</b>	(Optional) Primary terminal line.
<b>vty</b>	(Optional) Virtual terminal line.
<i>linenum</i>	(Optional) Line number. The number ranges from 0 to 999, depending on what type of line is selected.

### Usage Guidelines

Use the **debug arap** command with the **debug callback** command on access servers to debug dial-in and callback events.

### Sample Display

Figure 2-27 shows sample **debug arap internal** output.

**Figure 2-27 Sample Debug ARAP Internal Output**

```
Router# debug arap internal

ARAP: ----- SRVVERSION -----
ARAP: ----- ACKing 0 -----
ARAP: ----- AUTH_CHALLENGE -----
arapsec_local_account setting up callback
ARAP: ----- ACKing 1 -----
ARAP: ----- AUTH_RESPONSE -----
arap_startup initiating callback ARAP 2.0
ARAP: ----- CALLBACK -----
TTY7 Callback process initiated, user: dialback dialstring 40
TTY7 Callback forced wait = 4 seconds
TTY7 ARAP Callback Successful - await exec/autoselect pickup
TTY7: Callback in effect
ARAP: ----- STARTINFOFROMSERVER -----
ARAP: ----- ACKing 0 -----
ARAP: ----- ZONELISTINFO -----
ARAP: ----- ZONELISTINFO -----
ARAP: ----- ZONELISTINFO -----
```

```
ARAP: ----- ZONELISTINFO -----  
ARAP: ----- ZONELISTINFO -----
```

The displayed information is self-explanatory.

#### Related Command

**debug callback**

## 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.

**[no] debug arp**

### 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 2-28 shows sample **debug arp** output.

**Figure 2-28 Sample Debug ARP Output**

```
Router# debug arp

IP ARP: sent req src 172.16.22.7 0000.0c01.e117, dst 172.16.22.96 0000.0000.0000
IP ARP: rcvd rep src 172.16.22.96 0800.2010.b908, dst 172.16.22.7
IP ARP: rcvd req src 172.16.6.10 0000.0c00.6fa2, dst 172.16.6.62
IP ARP: rep filtered src 172.16.22.7 aa92.1b36.a456, dst 255.255.255.255 ffff.ffff.ffff
IP ARP: rep filtered src 172.16.9.7 0000.0c00.6b31, dst 172.16.22.7 0800.2010.b908
```

In Figure 2-28, 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 172.16.22.7 and MAC address 0000.0c01.e117 sent an ARP request for the MAC address of the host at 172.16.22.96. 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 172.16.22.7 0000.0c01.e117, dst 172.16.22.96 0000.0000.0000
```

The second line indicates that the router at IP address 172.16.22.7 receives a reply from the host at 172.16.22.96 indicating that its MAC address is 0800.2010.b908:

```
IP ARP: rcvd rep src 172.16.22.96 0800.2010.b908, dst 172.16.22.7
```

The third line indicates that the router receives an ARP request from the host at 172.16.6.10 requesting the MAC address for the host at 172.16.6.62:

```
IP ARP: rcvd req src 172.16.6.10 0000.0c00.6fa2, dst 172.16.6.62
```

The fourth line indicates that another host on the network attempted to send the router an ARP reply for its own address. The router ignores meaningless replies. Usually, meaningless replies happen if someone is running a bridge in parallel with the router and is allowing ARP to be bridged. This condition indicates a network misconfiguration.

```
IP ARP: rep filtered src 172.16.22.7 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 172.16.9.7, but the router does not know that the 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 172.16.9.7 0000.0c00.6b31, dst 172.16.22.7 0800.2010.b908
```

## debug asp packet

Use the **debug asp packet** EXEC command to display information on all asynchronous security protocols operating on the router. The **no** form of this command disables debugging output.

**[no] debug asp packet**

### Usage Guidelines

The router uses asynchronous security protocols such as ADT Security Systems, Inc., Adplex, and Diebold to transport alarm blocks between two devices (such as a security alarm system console and an alarm panel). The alarm blocks are transported in passthrough mode using BSTUN encapsulation.

### Sample Display

Figure 2-29 shows a partial sample **debug asp packet** output for asynchronous security protocols when packet debugging is enabled on an asynchronous line carrying Diebold alarm traffic. In this example, two polls are sent from the Diebold alarm console to two alarm panels that are multi-dropped from a single RS-232 interface. The alarm panels have device addresses F0 and F1. The example trace indicates that F1 is responding and F0 is not responding. At this point, you need to examine the physical link and possibly use a datascop to determine why the device is not responding.

**Figure 2-29 Sample Debug ASP Packet Output**

```
Router# debug asp packet

12:19:48: ASP: Serial5: ADI-Rx: Data (4 bytes): F1FF4C42
12:19:49: ASP: Serial5: ADI-Tx: Data (1 bytes): 88
12:19:49: ASP: Serial5: ADI-Rx: Data (4 bytes): F0FF9B94
12:20:47: ASP: Serial5: ADI-Rx: Data (4 bytes): F1FF757B
12:20:48: ASP: Serial5: ADI-Tx: Data (1 bytes): F3
12:20:48: ASP: Serial5: ADI-Rx: Data (4 bytes): F0FFB1BE
12:21:46: ASP: Serial5: ADI-Rx: Data (4 bytes): F1FFE6E8
12:21:46: ASP: Serial5: ADI-Tx: Data (1 bytes): 6F
12:21:46: ASP: Serial5: ADI-Rx: Data (4 bytes): F0FFC1CE
```

Table 2-19 describes the fields and messages shown in Figure 2-29.

**Table 2-19 Debug ASP Packet Field Descriptions**

Field	Description
ASP	Async security protocol packet.
Serial 5	Interface receiving and transmitting the packet.
ADI-Rx	Packet is being received.
ADI-TX	Packet is being transmitted.
Data ( <i>n</i> bytes)	Type and size of the packet.
F1FF4c42	Alarm panel device address.

## 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.

**[no] debug atm errors**

### Sample Display

Figure 2-30 shows sample **debug atm errors** output.

**Figure 2-30 Sample Debug ATM Errors Output**

```
Router# debug atm errors  
  
ATM(ATM2/0): Encapsulation error, link=7, host=836CA86D.  
ATM(ATM4/0): VCD#7 failed to echo OAM. 4 tries
```

The first line of output in Figure 2-30 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.

The second line of output shows that an OAM F5 (virtual circuit) cell error occurred.

## debug atm events

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

**[no] debug atm events**

### 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 2-31 shows sample **debug atm events** output.

**Figure 2-31 Sample Debug ATM Events Output**

```
Router# debug atm events

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 2-20 describes significant fields in the output shown in Figure 2-31.

**Table 2-20 Debug ATM Events Field Descriptions**

Field	Description
PLIM type	Indicates the interface rate in Mbps. Possible values are <ul style="list-style-type: none"> <li>• 1 = TAXI(4B5B) 100 Mbps</li> <li>• 2 = SONET 155 Mbps</li> <li>• 3 = E3 34 Mbps</li> </ul>
state	Indicates current state of the AIP. Possible values are <ul style="list-style-type: none"> <li>• 1 = An ENABLE will be issued soon</li> <li>• 0 = The AIP will remain shut down</li> </ul>
asr	Defines a bitmask, which indicates actions or completions to commands. Valid bitmask values are <ul style="list-style-type: none"> <li>• 0x0800 = AIP crashed, reload may be required.</li> <li>• 0x0400 = AIP detected a carrier state change.</li> <li>• 0x0n00 = Command completion status. Command completion status codes are <ul style="list-style-type: none"> <li>— n = 8 Invalid PLIM detected</li> <li>— n = 4 Command failed</li> <li>— n = 2 Command completed successfully</li> <li>— n = 1 CONFIG request failed</li> <li>— n = 0 Invalid value</li> </ul> </li> </ul>

Explanations for representative lines of output in Figure 2-31 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 oam

Use the **debug atm oam** EXEC command to display ATM operation and maintenance (OAM) events. The **no** form of this command disables debugging output.

**[no] debug atm oam**

### Sample Display

Figure 2-32 shows sample **debug atm oam** output.

**Figure 2-32 Sample Debug ATM OAM Output**

```
Router# debug atm oam

ATM4/0(O): VCD:0x0 DM:0x300 *OAM Cell* Length:0x39
0000 0300 0070 007A 0018 0100 0000 05FF FFFF FFFF FFFF FFFF FFFF FFFF FFFF
FFFF FFFF FFFF FFFF FF6A 6A6A 6A6A 6A6A 6A6A 6A6A 6A6A 6A6A 6A00 0000
```

Table 2-21 describes the output fields shown in Figure 2-32.

**Table 2-21 Debug ATM OAM Field Descriptions**

Field	Description
0000	VCD Special OAM indicator
0300	Descriptor MODE bits for the AIP
0	GFC (4 bits)
07	VPI (8 bits)
0007	VCI (16 bits)
A	Payload type field(PTI)(4 bits)
00	Header Error Correction(8 bits)
1	OAM Fault mgmt. cell(4 bits)
8	OAM LOOPBACK indicator (4 bits)
01	Loopback indicator value, always 1(8 bits)
00000005	Loopback unique ID, sequence number (32 bits)
FF6A	F's and 6A required in the remaining ATM cell, per UNI3.0

## 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.

**[no] debug atm packet [interface atm number [vcd vcd-number]]**

### Syntax Description

**interface** *number* (Optional) ATM interface or subinterface number.

**vcd** *vcd-number* (Optional) Number of the virtual circuit designator (VCD).

### 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 2-33 shows sample **debug atm packet** output.

**Figure 2-33 Sample Debug ATM Packet Output**

```
Router# debug atm packet

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 DCFE 0105
```

Table 2-22 describes significant fields shown in Figure 2-33.

**Table 2-22 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: 0xn	Indicates the virtual circuit associated with this packet, where <i>n</i> is some value.
DM: 0xnnnn	Indicates the descriptor mode bits on output only, where <i>nnnn</i> is a hexadecimal value.
ETYPE: <i>n</i>	Shows the Ethernet type for this packet.
Length: <i>n</i>	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 DCFE 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.

**[no] debug bri**

### 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 2-34 shows sample **debug bri** output.

**Figure 2-34 Sample Debug BRI Output**

```
Router# debug bri

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 2-34 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 bsc event

Use the **debug bsc event** EXEC command to display all events occurring in the Binary Synchronous Communications (BSC) feature. The **no** form of this command disables debugging output.

**[no] debug bsc event** *[number]*

### Syntax Description

*number* (Optional) Group number.

### Usage Guidelines

This command traces all interfaces configured with a **bsc protocol-group** *number* command.

### Sample Display

Figure 2-35 shows sample **debug bsc event** output.

**Figure 2-35 Sample Debug BSC Event Output**

```
Router# debug bsc event

BSC: Serial2          POLLEE-FSM inp:E_LineFail old_st:CU_Down new_st:TCU_EOFFile
BSC: Serial2          POLLEE-FSM inp:E_LineFail old_st:CU_Down new_st:TCU_EOFFile
BSC: Serial2          POLLEE-FSM inp:E_LineFail old_st:CU_Down new_st:TCU_EOFFile
0:04:32: BSC: Serial2 :SDI-rx: 9 bytes
BSC: Serial2          POLLEE-FSM inp:E_RxEtx old_st:CU_Down new_st:TCU_EOFFile
0:04:32: BSC: Serial2 :SDI-rx: 5 bytes
BSC: Serial2          POLLEE-FSM inp:E_RxEng old_st:CU_Down new_st:TCU_EOFFile
BSC: Serial2          POLLEE-FSM inp:E_Timeout old_st:CU_Down new_st:TCU_InFile
BSC: Serial2          POLLEE-FSM inp:E_Timeout old_st:CU_Idle new_st:TCU_InFile
%LINEPROTO-5-UPDOWN: Line protocol on Interface Serial2, changed state to up
%LINK-3-UPDOWN: Interface Serial2, changed state to up
BSC: Serial2          POLLEE-FSM inp:E_Timeout old_st:CU_Idle new_st:TCU_InFile
0:04:35: BSC: Serial2 :SDI-rx: 9 bytes
BSC: Serial2          POLLEE-FSM inp:E_RxEtx old_st:CU_Idle new_st:TCU_InFile
0:04:35: BSC: Serial2 :SDI-rx: 5 bytes
BSC: Serial2          POLLEE-FSM inp:E_RxEng old_st:CU_Idle new_st:TCU_InFile
0:04:35: BSC: Serial2 :NDI-rx: 3 bytes
```

### Related Commands

**debug bsc packet**

**debug bstun events**

## debug bsc packet

Use the **debug bsc packet** EXEC command to display all frames traveling through the Binary Synchronous Communications (BSC) feature. The **no** form of this command disables debugging output.

```
[no] debug bsc packet [group number] [buffer-size bytes]
```

### Syntax Description

<b>group number</b>	(Optional) Group number.
<b>buffer-size bytes</b>	(Optional) Number of bytes displayed per packet (defaults to 20).

### Usage Guidelines

This command traces all interfaces configured with a **bsc protocol-group number** command.

### Sample Display

Figure 2-36 shows sample **debug bsc packet** output.

**Figure 2-36 Sample Debug BSC Packet Output**

```
Router# debug bsc packet
0:23:33: BSC: Serial2      :NDI-rx : 27 bytes 401A400227F5C31140C11D60C8C5D3D3D51D4013
0:23:33: BSC: Serial2      :SDI-tx  : 12 bytes 00323237FF3232606040402D
0:23:33: BSC: Serial2      :SDI-rx  : 2 bytes 1070
0:23:33: BSC: Serial2      :SDI-tx  : 27 bytes 401A400227F5C31140C11D60C8C5D3D3D51D4013
0:23:33: BSC: Serial2      :SDI-rx  : 2 bytes 1061
0:23:33: BSC: Serial2      :SDI-tx  : 5 bytes 00323237FF
```

### Related Commands

**debug bsc event**  
**debug bstun events**

## debug bstun events

Use the **debug bstun events** EXEC command to display BSTUN connection events and status. The **no** form of this command disables debugging output.

**[no] debug bstun events** *number*

### Syntax Description

*number* (Optional) Group number.

### Usage Guidelines

When you enable the **debug bstun events** command, messages showing connection establishment and other overall status messages are displayed.

You can use the **debug bstun events** command to assist you in determining whether the BSTUN peers are configured correctly and are communicating. For example, if you enable the **debug bstun packet** command and you do not see any packets, you may want to enable event debugging.

---

**Note** Also refer to the **debug bsc packet** and **debug bsc event** commands. Currently, these two commands support the only protocol working through the BSTUN tunnel. Sometimes frames do not go through the tunnel because they have been discarded at the BSC protocol level.

---

### Sample Displays

Figure 2-37 shows a sample **debug bstun events** output of keepalive messages working correctly. If the routers are configured correctly, at least one router will show reply messages.

**Figure 2-37 Sample Debug BSTUN Events Output—Keepalive Messages**

```
Router# debug bstun packet

BSTUN: Received Version Reply opcode from (all[2])_172.16.12.2/1976 at 1360
BSTUN: Received Version Request opcode from (all[2])_172.16.12.2/1976 at 1379
BSTUN: Received Version Reply opcode from (all[2])_172.16.12.2/1976 at 1390
```

---

**Note** In a scenario where there is constantly loaded bi-directional traffic, you might not see keepalive messages because they are sent only when the remote end has been silent for the keepalive period.

---

Figure 2-38 shows a sample **debug bstun events** output of an event trace in which the wrong TCP address has been specified for the remote peer. These are non-keepalive related messages.

**Figure 2-38 Sample Debug BSTUN Events Output—Event Trace**

```
Router# debug bstun packet

BSTUN: Change state for peer (C1[1])172.16.12.22/1976 (closed->opening)
BSTUN: Change state for peer (C1[1])172.16.12.22/1976 (opening->open wait)
%BSTUN-6-OPENING: CONN: opening peer (C1[1])172.16.12.22/1976, 3
BSTUN: tcpd sender in wrong state, dropping packet
BSTUN: tcpd sender in wrong state, dropping packet
BSTUN: tcpd sender in wrong state, dropping packet
```

### Related Commands

**debug bsc event**

**debug bsc packet**

**debug bstun packet**

## debug bstun packet

Use the **debug bstun packet** EXEC command to display packet information on packets traveling through the BSTUN links. The **no** form of this command disables debugging output.

**[no] debug bstun packet [group number] [buffer-size bytes]**

### Syntax Description

<b>group number</b>	(Optional) BSTUN group number.
<b>buffer-size bytes</b>	(Optional) Number of bytes displayed per packet (defaults to 20).

### Sample Display

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

**Figure 2-39 Sample Debug BSTUN Packet Output**

```
Router# debug bstun packet
BSTUN bsc-local-ack: 0:00:00 Serial2      SDI: Addr: 40 Data: 02C1C1C1C1C1C1C1C1
BSTUN bsc-local-ack: 0:00:00 Serial2      SDI: Addr: 40 Data: 02C1C1C1C1C1C1C1C1
BSTUN bsc-local-ack: 0:00:06 Serial2      NDI: Addr: 40 Data: 0227F5C31140C11D60C8
```

### Related Command

**debug bstun events**

## debug callback

Use the **debug callback EXEC** command to display callback events when the router is using a modem and a chat script to call back on a terminal line. The **no** form of this command disables debugging output.

**[no] debug callback**

### Usage Guidelines

This command is useful for debugging chat scripts on PPP and ARAP lines that use callback mechanisms. The output provided by the **debug callback** command shows you how the call is progressing when used with the **debug ppp** or **debug arap** commands.

### Sample Display

Figure 2-40 shows sample **debug callback** output.

**Figure 2-40 Sample Debug Callback Output**

```
Router# debug callback

TTY7 Callback process initiated, user: exec_test dialstring 123456
TTY7 Callback forced wait = 4 seconds
TTY7 Exec Callback Successful - await exec/autoselect pickup
TTY7: Callback in effect
```

### Related Commands

**debug arap**  
**debug ppp**

## 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.

```
[no] debug cdp {packets | adjacency | events}
```

### Syntax Description

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

### Usage Guidelines

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

### Sample Display

Figure 2-41 shows a composite sample output from **debug cdp packets**, **debug cdp adjacency**, and **debug cdp events**.

**Figure 2-41 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 cdp ip

Use the **debug cdp ip** EXEC command to enable debug output for the IP routing information that is carried and processed by the Cisco Discovery Protocol (CDP). The **no** form of this command disables debugging output.

**[no] debug cdp ip**

### Usage Guidelines

CDP is a media- and protocol-independent device-discovery protocol that runs on all Cisco routers.

You can use the **debug cdp ip** command to determine the IP network prefixes CDP is advertising and whether CDP is correctly receiving this information from neighboring routers.

Use the **debug cdp ip** command with the **debug ip routing** command to debug problems that occur when on-demand routing (ODR) routes are not installed in the routing table at a hub router. You can also use the **debug cdp ip** command with the **debug cdp packet** and **debug cdp adjacency** commands along with encapsulation-specific debug commands to debug problems that occur in the receipt of CDP IP information.

### Sample Display

Figure 2-42 shows sample **debug cdp ip** output. This example shows the transmission of IP-specific information in a CDP update. In this case, three network prefixes are being transmitted, each with a different network mask.

**Figure 2-42 Sample Debug CDP IP Output**

```
Router# debug cdp ip

CDP-IP: Writing prefix 172.1.69.232.112/28
CDP-IP: Writing prefix 172.19.89.0/24
CDP-IP: Writing prefix 11.0.0.0/8
```

In addition to the messages shown in Figure 2-42, you might see the following messages:

- This message indicates that CDP is attempting to install the prefix 172.1.1.0/24 into the IP routing table:  

```
CDP-IP: Updating prefix 172.1.1.0/24 in routing table
```
- This message indicates a protocol error occurred during an attempt to decode an incoming CDP packet:  

```
CDP-IP: IP TLV length (3) invalid
```
- This message indicates the receipt of the IP prefix 172.1.1.0/24 from a CDP neighbor connected via the Ethernet interface 0/0. The neighbor's IP address is 10.0.0.1.  

```
CDP-IP: Reading prefix 172.1.1.0/24 source 10.0.0.1 via Ethernet0/0
```

### Related Commands

**debug cdp adjacency**  
**debug cdp packet**  
**debug ip routing**

## 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.

**[no] debug channel events**

### 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. If the command generates numerous messages, they can indicate the possible source of the problems. To observe the statistic message (cip\_love\_letter) transmitted every ten seconds, use the **debug channel love** command.

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 2-43 shows sample **debug channel events** output.

**Figure 2-43 Sample Debug Channel Events Output**

```
Router# debug channel events

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 2-43 follow.

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 virtual circuit number 0 with this device. The virtual circuit number will also show up when you use the **debug channel packets** command.

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

Related Commands

**debug channel love**

**debug channel packets**

## debug channel love

Use the **debug channel love** EXEC command to display Channel Interface Processor (CIP) love letter events. This command is valid for the Cisco 7000 series routers only. The **no** form of this command disables debugging output.

**[no] debug channel love**

### 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 love** command returns a statistic message (cip\_love\_letter) that is transmitted every ten seconds.

### Sample Display

Figure 2-44 shows sample **debug channel love** output.

**Figure 2-44 Sample Debug Channel Love Output**

```
Router# debug channel love

Channel3/1: love letter received, bytes 3308
Channel3/0: love letter received, bytes 3336
cip_love_letter: received 11, but no cip_info
```

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 11, but no cip_info
```

### Related Commands

**debug channel events**

**debug channel packets**

## 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.

**[no] debug channel packets**

### Usage Guidelines

The **debug channel packets** command displays all process-level Channel Interface Processor (CIP) packets for both outbound 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 2-45 shows sample **debug channel packets** output.

**Figure 2-45 Sample Debug Channel Packets Output**

```
Router# debug channel packets

(Channel3/0)-out size = 104, vc = 0000, type = 0800, src 172.24.0.11, dst 172.24.1.58
(Channel3/0)-in size = 48, vc = 0000, type = 0800, src 172.24.1.58, dst 172.24.15.197
(Channel3/0)-in size = 48, vc = 0000, type = 0800, src 172.24.1.58, dst 172.24.15.197
(Channel3/0)-out size = 71, vc = 0000, type = 0800, src 172.24.15.197, dst 172.24.1.58
(Channel3/0)-in size = 44, vc = 0000, type = 0800, src 172.24.1.58, dst 172.24.15.197
```

Table 2-23 provides explanations for individual lines of output from Figure 2-45.

**Table 2-23 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 <b>claw</b> interface configuration command. This information is from the MAC layer.
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.

### Related Commands

**debug channel events**  
**debug channel love**