

debug ip pim

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

[no] debug ip pim [group]

Syntax Description

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

Usage Guidelines

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

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

Sample Display

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

```
Router# debug ip pim 224.2.0.1

PIM: Received Join/Prune on Ethernet1 from 172.24.37.33
PIM: Received Join/Prune on Ethernet1 from 172.24.37.33
PIM: Received Join/Prune on Tunnel0 from 10.3.84.1
PIM: Received Join/Prune on Ethernet1 from 172.24.37.33
PIM: Received Join/Prune on Ethernet1 from 172.24.37.33
PIM: Received RP-Reachable on Ethernet1 from 172.16.20.31
PIM: Update RP expiration timer for 224.2.0.1
PIM: Forward RP-reachability packet for 224.2.0.1 on Tunnel0
PIM: Received Join/Prune on Ethernet1 from 172.24.37.33
PIM: Prune-list (10.221.196.51/32, 224.2.0.1)
PIM: Set join delay timer to 2 seconds for (10.221.0.0/16, 224.2.0.1) on Ethernet1
PIM: Received Join/Prune on Ethernet1 from 172.24.37.6
PIM: Received Join/Prune on Ethernet1 from 172.24.37.33
PIM: Received Join/Prune on Tunnel0 from 10.3.84.1
PIM: Join-list: (*, 224.2.0.1) RP 172.16.20.31
PIM: Add Tunnel0 to (*, 224.2.0.1), Forward state
PIM: Join-list: (10.0.0.0/8, 224.2.0.1)
PIM: Add Tunnel0 to (10.0.0.0/8, 224.2.0.1), Forward state
PIM: Join-list: (10.4.0.0/16, 224.2.0.1)
PIM: Prune-list (172.24.84.16/28, 224.2.0.1) RP-bit set RP 172.24.84.16
PIM: Send Prune on Ethernet1 to 172.24.37.6 for (172.24.84.16/28, 224.2.0.1), RP
PIM: For RP, Prune-list: 10.9.0.0/16
PIM: For RP, Prune-list: 10.16.0.0/16
PIM: For RP, Prune-list: 10.49.0.0/16
PIM: For RP, Prune-list: 10.84.0.0/16
PIM: For RP, Prune-list: 10.146.0.0/16
PIM: For 10.3.84.1, Join-list: 172.24.84.16/28
PIM: Send periodic Join/Prune to RP via 172.24.37.6 (Ethernet1)
```

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

```
PIM: Received Join/Prune on Ethernet1 from 172.24.37.33
PIM: Received Join/Prune on Ethernet1 from 172.24.37.33
```

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

```
PIM: Received RP-Reachable on Ethernet1 from 172.16.20.31
PIM: Update RP expiration timer for 224.2.0.1
PIM: Forward RP-reachability packet for 224.2.0.1 on Tunnel0
```

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

```
PIM: Prune-list (10.221.196.51/32, 224.2.0.1)
```

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

```
PIM: Set join delay timer to 2 seconds for (10.221.0.0/16, 224.2.0.1) on Ethernet1
```

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

```
PIM: Join-list: (*, 224.2.0.1) RP 172.16.20.31
```

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

```
PIM: Add Tunnel0 to (*, 224.2.0.1), Forward state
PIM: Add Tunnel0 to (10.0.0.0/8, 224.2.0.1), Forward state
```

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

```
PIM: Prune-list (172.24.84.16/28, 224.2.0.1) RP-bit set RP 172.24.84.16
```

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

```
PIM: Send Prune on Ethernet1 to 172.24.37.6 for (172.24.84.16/28, 224.2.0.1), RP
```

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

```
PIM: For RP, Prune-list: 10.9.0.0/16
PIM: For RP, Prune-list: 10.16.0.0/16
PIM: For RP, Prune-list: 10.49.0.0/16
```

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

```
PIM: Send periodic Join/Prune to RP via 172.24.37.6 (Ethernet1)
```

Related Commands

debug ip dvmrp
debug ip igmp
debug ip igmp transactions
debug ip mrouting
debug ip sd

debug ip pim atm

To log PIM ATM signaling activity, use the **debug ip pim atm EXEC** command. The **no** form of this command disables debugging output.

[no] debug ip pim atm

Sample Displays

The following is sample output from the shows a new group being created and the router toward the RP opening a new VC. Since there are now two groups on this router, there are two virtual circuits open, as reflected by the “current count.”

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

```
Router# debug ip pim atm

Jan 28 19:05:51: PIM-ATM: Max VCs 200, current count 1
Jan 28 19:05:51: PIM-ATM: Send SETUP on ATM2/0 for 239.254.254.253/171.69.214.43
Jan 28 19:05:51: PIM-ATM: Received CONNECT on ATM2/0 for 239.254.254.253, vcd 19
Jan 28 19:06:35: PIM-ATM: Max VCs 200, current count 2
```

Table 66 describes the significant fields in the output.

Table 66 Debug IP PIM ATM Field Descriptions

Field	Description
Jan 28 19:05:51	Current date and time in hours:minutes:seconds.
PIM-ATM	Indicates what PIM is doing to set up or monitor an ATM connection (vc).
current count	Current number of open virtual circuits.

The resulting **show ip mroute** output follows:

```
Router# show ip mroute 239.254.254.253

IP Multicast Routing Table
Flags: D - Dense, S - Sparse, C - Connected, L - Local, P - Pruned
      R - RP-bit set, F - Register flag, T - SPT-bit set, J - Join SPT
Timers: Uptime/Expires
Interface state: Interface, Next-Hop or VCD, State/Mode

(*, 239.254.254.253), 00:00:04/00:02:53, RP 171.69.214.50, flags: S
  Incoming interface: Ethernet1/1, RPF nbr 171.69.214.50
  Outgoing interface list:
    ATM2/0, VCD 19, Forward/Sparse-Dense, 00:00:04/00:02:52
```

debug ip pim auto-rp

Use the **debug ip pim auto-rp EXEC** command to display the contents of each Protocol Independent Multicast (PIM) packet used in the automatic discovery of group-to-rendezvous point (RP) mapping as well as the actions taken on the address-to-RP mapping database. The **no** form of this command disables debugging output.

[no] debug ip pim auto-rp

Sample Displays

The following is sample output from the **debug ip pim auto-rp** command:

```
Router# debug ip pim auto-rp

Auto-RP: Received RP-announce, from 172.31.214.66, RP_cnt 1, holdtime 180 secs
Auto-RP:  update (192.168.248.0/24, RP:172.31.214.66)
Auto-RP: Build RP-Discovery packet
Auto-RP:  Build mapping (192.168.248.0/24, RP:172.31.214.66),
Auto-RP:  Build mapping (192.168.250.0/24, RP:172.31.214.26).
Auto-RP:  Build mapping (192.168.254.0/24, RP:172.31.214.2).
Auto-RP: Send RP-discovery packet (3 RP entries)
Auto-RP: Build RP-Announce packet for 172.31.214.2
Auto-RP:  Build announce entry for (192.168.254.0/24)
Auto-RP: Send RP-Announce packet, IP source 172.31.214.2, ttl 8
```

The first two lines show a packet received from 172.31.214.66 announcing that it is the rendezvous point (RP) for the groups in 192.168.248.0/24. This announcement contains one RP address and is valid for 180 seconds. The RP-mapping agent then updates its mapping database to include the new information.

```
Auto-RP: Received RP-announce, from 172.31.214.66, RP_cnt 1, holdtime 180 secs
Auto-RP:  update (192.168.248.0/24, RP:172.31.214.66)
```

In the next five lines, the router creates an RP-discovery packet containing three RP mapping entries. The packet is sent to the well-known CISCO-RP-DISCOVERY group address (224.0.1.40).

```
Auto-RP: Build RP-Discovery packet
Auto-RP:  Build mapping (192.168.248.0/24, RP:172.31.214.66),
Auto-RP:  Build mapping (192.168.250.0/24, RP:172.31.214.26).
Auto-RP:  Build mapping (192.168.254.0/24, RP:172.31.214.2).
Auto-RP: Send RP-discovery packet (3 RP entries)
```

The final three lines show the router announcing that it intends to be an RP for the groups in 192.168.254.0/24. Only routers inside the scope ttl 8 receive the advertisement and use the RP for these groups.

```
Auto-RP: Build RP-Announce packet for 172.31.214.2
Auto-RP:  Build announce entry for (192.168.254.0/24)
Auto-RP: Send RP-Announce packet, IP source 172.31.214.2, ttl 8
```

The following is sample output from the **debug ip pim auto-rp** command when a router receives an update. In this example, the packet contains three group-to-RP mappings, which are valid for 180 seconds. The RP-mapping agent then updates its mapping database to include the new information.

```
Router# debug ip pim auto-rp
```

```
Auto-RP: Received RP-discovery, from 172.31.214.17, RP_cnt 3, holdtime 180 secs  
Auto-RP:  update (192.168.248.0/24, RP:172.31.214.66)  
Auto-RP:  update (192.168.250.0/24, RP:172.31.214.26)  
Auto-RP:  update (192.168.254.0/24, RP:172.31.214.2)
```

debug ip policy

Use the **debug ip policy** EXEC command to display IP policy routing packet activity. The **no** form of this command disables debugging output.

[no] debug ip policy

Usage Guidelines

After you configure IP policy routing with the **ip policy** and **route map** commands, use the **debug ip policy** command to ensure that the IP policy is configured correctly.

Policy routing looks at various parts of the packet and then routes the packet based on certain user-defined attributes in the packet.

The **debug ip policy** command helps you determine what policy routing is doing. It displays information about whether a packet matches the criteria, and if so, the resulting routing information for the packet.



Caution Because the **debug ip policy** 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.

You can also use the **show ip local policy** command to obtain additional information.

Sample Display

The following is sample output from the **debug ip policy** command. Line 1 indicates that a packet with the given source and destination addresses matched a policy. Line 2 indicates the clause in the route map that the packet matched. In this case, the packet matches clause 20 in the route map. Line 3 indicates that a second packet did not match the policy.

```
Router# debug ip policy

IP: s=172.16.232.150 (local), d=172.16.2.75, len 100, policy match
IP: route map equal, item 20, permit
IP: s=172.16.232.150 (local), d=172.16.2.75, len 200, policy rejected -- normal
forwarding
```

debug ip rip

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

[no] debug ip rip

Sample Display

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

```

router# debug ip rip
Updates
received  — RIP: received update from 10.89.80.28 on Ethernet0
from this  10.89.95.0 in 1 hops
source    10.89.81.0 in 1 hops
address   10.89.66.0 in 2 hops
          172.31.0.0 in 16 hops (inaccessible)
          0.0.0.0 in 7 hop
Updates
sent to   — RIP: sending update to 255.255.255.255 via Ethernet0 (10.89.64.31)
these two subnet 10.89.94.0, metric 1
destination 172.31.0.0 in 16 hops (inaccessible)
addresses  — RIP: sending update to 255.255.255.255 via Serial1 (10.89.94.31)
           subnet 10.89.64.0, metric 1
           subnet 10.89.66.0, metric 3
           172.31.0.0 in 16 hops (inaccessible)
           default 0.0.0.0, metric 8

```

S22550

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

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

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

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

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

```

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

```

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

```

RIP: bad version 128 from 160.89.80.43

```

debug ip routing

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

[no] debug ip routing

Sample Display

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

```
Router# debug ip routing

RT: add 172.25.168.0 255.255.255.0 via 172.24.76.30, igrp metric [100/3020]
RT: metric change to 172.25.168.0 via 172.24.76.30, igrp metric [100/3020]
    new metric [100/2930]
IP: cache invalidation from 0x115248 0x1378A, new version 5736
RT: add 172.26.219.0 255.255.255.0 via 172.24.76.30, igrp metric [100/16200]
RT: metric change to 172.26.219.0 via 172.24.76.30, igrp metric [100/16200]
    new metric [100/10816]
RT: delete route to 172.26.219.0 via 172.24.76.30, igrp metric [100/10816]
RT: no routes to 172.26.219.0, entering holddown
IP: cache invalidation from 0x115248 0x1378A, new version 5737
RT: 172.26.219.0 came out of holddown
RT: garbage collecting entry for 172.26.219.0
IP: cache invalidation from 0x115248 0x1378A, new version 5738
RT: add 172.26.219.0 255.255.255.0 via 172.24.76.30, igrp metric [100/10816]
RT: delete route to 172.26.219.0 via 172.24.76.30, igrp metric [100/10816]
RT: no routes to 172.26.219.0, entering holddown
IP: cache invalidation from 0x115248 0x1378A, new version 5739
RT: 172.26.219.0 came out of holddown
RT: garbage collecting entry for 172.26.219.0
IP: cache invalidation from 0x115248 0x1378A, new version 5740
RT: add 172.26.219.0 255.255.255.0 via 172.24.76.30, igrp metric [100/16200]
RT: metric change to 172.26.219.0 via 172.24.76.30, igrp metric [100/16200]
    new metric [100/10816]
RT: delete route to 172.26.219.0 via 172.24.76.30, igrp metric [100/10816]
RT: no routes to 172.26.219.0, entering holddown
IP: cache invalidation from 0x115248 0x1378A, new version 5741
```

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

```
RT: add 172.25.168.0 255.255.255.0 via 172.24.76.30, igrp metric [100/3020]
RT: metric change to 172.25.168.0 via 172.24.76.30, igrp metric [100/3020]
    new metric [100/2930]
IP: cache invalidation from 0x115248 0x1378A, new version 5736
```

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

In the following output, the “holddown” and “cache invalidation” lines are displayed. Most of the distance vector routing protocols use “holddown” to avoid typical problems like counting to infinity and routing loops. If you look at the output of **show ip protocols** you will see what the timer values

are for “holddown” and “cache invalidation.” “Cache invalidation” corresponds to “came out of holddown.” “Delete route” is triggered when a better path comes along. It gets rid of the old inferior path.

```
RT: delete route to 172.26.219.0 via 172.24.76.30, igrp metric [100/10816]
RT: no routes to 172.26.219.0, entering holddown
IP: cache invalidation from 0x115248 0x1378A, new version 5737
RT: 172.26.219.0 came out of holddown
```

debug ip rsvp

Use the **debug ip rsvp** EXEC command to enable logging of significant Resource Reservation Protocol (RSVP) events. The **no** form of this command disables debugging output.

[no] debug ip rsvp [detail [access-list]]

Syntax Description

detail	(Optional) Displays debug information in a detailed format.
<i>access-list</i>	(Optional) Standard IP access list number. If the datagram is not permitted by the specified access list, the related debugging output is suppressed.

Usage Guidelines

The RSVP protocol permits end systems to request quality of service (QoS) guarantees from the network.

The **debug ip rsvp** command displays recognition of new senders, subsequent discontinuation of senders, and installation and removal of reservations.

The **detail** option displays recognition of new senders, subsequent discontinuation of senders, installation and removal of reservations, and messages sent and received that are permitted by the specified access list, or all messages if no access list is specified. The output from **detail** option is the same as the **debug ip rsvp** command but it also includes a hexadecimal-dump of the contents of the messages.

Sample Displays

In the following message, the router received a path message for destination 192.168.253.10 on interface Ethernet 0/2. The message was sent by 172.31.215.9—this indicates the previous hop and not the source.

```
RSVP: PATH message for 192.168.253.10(Ethernet0/2) from 172.31.215.9
```

In the following two messages, the router is sending a path message to destination 192.168.253.10 on interface Ethernet 0/0. The second message is a reassurance that the packet is sent.

```
RSVP: send path multicast about 192.168.253.10 on Ethernet0/0
RSVP: IP to 192.168.253.10 length=52 checksum=720C Ethernet0/0
```

In the following message, the router received an reservation (RESV) message for destination 192.168.253.10 on interface serial 0. The message was sent by 172.31.215.113, which is the next hop toward the destination.

```
RSVP: RESV message for 192.168.253.10(Serial0) from 172.31.215.113
```

In the following messages, the router is removing the sender state for destination 1.1.1.1. The destination port is (1234), the source is 172.31.215.11, and the protocol and source port are (17:1234). Removing the sender state triggers a teardown message to be sent toward destination 1.1.1.1 on interface Ethernet 0/2 (located in the routing table).

```
RSVP: remove Ethernet0/1 PATH 1.1.1.1(1234) <- 172.31.215.11(17:1234)
RSVP: send path teardown multicast about 1.1.1.1 on Ethernet0/2
```

The following messages occur when a reservation (RESV) message is received but no sender state (path message) exists that matches it. In this case, the router drops the reservation and sends a reservation error message to destination 192.168.253.10.

```
RSVP: RESV message for 192.168.253.10(Serial0) from 172.31.215.113
RSVP: send reservation error to 172.31.215.113 about 192.168.253.10
RSVP: IP to 172.31.215.113 length=40 checksum=1A84 if Serial0
RSVP RESV: no path information for 192.168.253.10
```

In the following messages, the router tears down a reservation because it received a teardown message or because the message timed out. Tearing down a reservation triggers the removal of the existing installed reservation, as shown in the second message. The teardown also triggers the removal of this reservation from upstream routers. After the router removes the reservation, an RESV teardown message is sent to the source to tear down the reservation on the next hop, as shown in the fourth message. This process continues until the source is reached.

```
RSVP: RESV TEAR message for 192.168.253.10(Ethernet0/0) from 172.31.215.98
RSVP: remove Ethernet0/0 RESV 192.168.253.10(18004) <- 172.31.60.189(17:18004)
RSVP: remove Ethernet0/2 RESV request 192.168.253.10(18004) <- 172.31.60.189(17:18004)
RSVP: send reservation teardown to 172.31.215.9 about 192.168.253.10
```

In the following messages, the router received and accepted a new reservation request. This request triggers the router to send a reservation request message to the source.

```
RSVP: Reservation is new
RSVP: start requesting 30 kbps SE reservation for 172.31.60.189(18004) UDP->
192.168.253.10(18004) on Ethernet0 neighbor 172.31.215.97
```

In the following message, the router received an RSVP message on an interface with the RSVP feature disabled:

```
RSVP: Input packet while RSVP disabled on Serial2/0
```

debug ip rtp header-compression

Use the **debug ip rtp header-compression** EXEC command to display events specific to RTP header compression. Use the **no** form of this command to disable debugging output.

[no] debug ip rtp header-compression

Sample Display

The following is sample output from the **debug ip rtp header-compression** command:

```
Router# debug ip rtp header-compression

RHC BRI0: rcv compressed rtp packet
RHC BRI0: context0: expected sequence 0, received sequence 0
RHC BRI0: rcv compressed rtp packet
RHC BRI0: context0: expected sequence 1, received sequence 1
RHC BRI0: rcv compressed rtp packet
RHC BRI0: context0: expected sequence 2, received sequence 2
RHC BRI0: rcv compressed rtp packet
RHC BRI0: context0: expected sequence 3, received sequence 3
```

Table 67 describes the significant fields in the output.

Table 67 Debug IP RTP Header-Compression Field Descriptions

Field	Description
context 0	Compression state for a connection 0.
expected sequence	RTP header compression link sequence (expected).
received sequence	RTP header compression link sequence (actually received).

Related Command

debug ip rtp packets

debug ip rtp packets

Use the **debug ip rtp packets** EXEC command to display a detailed dump of packets specific to RTP header compression. Use the **no** form of this command to disable debugging output.

[no] debug ip rtp packets

Sample Display

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

```
Router# debug ip rtp packets

RTP packet dump:
  IP:  source: 171.68.8.10, destination: 224.2.197.169, id: 0x249B, ttl: 9,
      TOS: 0 prot: 17,
  UDP: source port: 1034, destination port: 27404, checksum: 0xB429, len: 152
  RTP: version: 2, padding: 0, extension: 0, marker: 0,
      payload: 3, ssrc 2369713968,
      sequence: 2468, timestamp: 85187180, csrc count: 0
```

Table 68 describes the significant fields in the output.

Table 68 Debug IP RTP Packets Field Descriptions

Field	Description
id	IP identification.
ttl	IP time to live (TTL).
len	Total UDP length.

Related Command

debug ip rtp header-compression

debug ip sd

Use the **debug ip sd** command to display all session directory (SD) announcements received. The **no** form of this command disables debugging output.

[no] debug ip sd

Usage Guidelines

This command shows session directory announcements for multicast IP. Use it to observe multicast activity.

Sample Display

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

```
Router# debug ip sd

SD: Announcement from 172.31.58.81 on Serial0.1, 146 bytes
s=*cisco: CBONE Audio
i=cisco internal-only audio conference
o=dino@dino-ss20.cisco.com
c=224.0.255.1 16 2891478496 2892688096
m=audio 31372 1700

SD: Announcement from 172.22.246.68 on Serial0.1, 147 bytes
s=IMS: U.S. Senate
i=U.S. Senate at http://town.hall.org/radio/live.html
o=carl@also.radio.com
c=224.2.252.231 95 0 0
m=audio 36572 2642
a=fmt:gsm
```

Table 69 provides explanations for representative lines of the **debug ip sd** output.

Table 69 Debug IP SD Output Descriptions

Field	Description
SD	Session directory event.
Announcement from	Address sending the SD announcement.
on Serial0.1	Interface receiving the announcement.
146 bytes	Size of the announcement event.
s=	Session name being advertised.
i=	Information providing a descriptive name for the session.
o=	Origin of the session, either an IP address or a name.
c=	Connect description showing address and number of hops.
m=	Media description that includes media type, port number, and ID.

Related Commands

debug ip dvmrp
debug ip igmp
debug ip mbgp dampening
debug ip mrouting
debug ip pim

debug ip security

Use the **debug ip security** EXEC command to display IP security option processing. The **no** form of this command disables debugging output.

[no] debug ip security

Usage Guidelines

The **debug ip security** command displays information for both basic and extended IP security options. For interfaces where **ip security** is configured, each IP packet processed for that interface results in debugging output regardless of whether the packet contains IP security options. IP packets processed for other interfaces that also contain IP security information also trigger debugging output. Some additional IP security debugging information is also controlled by the **debug ip packet** EXEC command.

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

Sample Display

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

```
Router# debug ip security

IP Security: src 172.24.72.52 dst 172.24.72.53, number of BSO 1
  idb: NULL
  pak: insert (0xFF) 0x0
IP Security: BSO postroute: SECINSERT changed to secret (0x5A) 0x10
IP Security: src 172.24.72.53 dst 172.24.72.52, number of BSO 1
  idb: secret (0x6) 0x10 to secret (0x6) 0x10, no implicit
  def secret (0x6) 0x10
  pak: secret (0x5A) 0x10
IP Security: checking BSO 0x10 against [0x10 0x10]
IP Security: classified BSO as secret (0x5A) 0x10
```

Table 70 describes significant fields in the output.

Table 70 Debug IP Security Field Descriptions

Field	Description
number of BSO	Indicates the number of basic security options found in the packet.
idb	Provides information on the security configuration for the incoming interface.
pak	Provides information on the security classification of the incoming packet.
src	Indicates the source IP address.
dst	Indicates the destination IP address.

The following line indicates that the packet was locally generated, and it has been classified with the internally significant security level “insert” (0xff) and authority 0x0:

```
idb: NULL
pak: insert (0xff) 0x0
```

The following line indicates that the packet was received via an interface with dedicated IP security configured. Specifically, the interface is configured at security level “secret” and with authority information of 0x0. The packet itself was classified at level “secret” (0x5a) and authority 0x10.

```
idb: secret (0x6) 0x10 to secret (0x6) 0x10, no implicit
     def secret (0x6) 0x10
pak: secret (0x5A) 0x10
```

debug ip socket

Use the **debug ip socket EXEC** command to display all state change information for all sockets. Use the **no** form of this command to disable debugging output.

[no] debug ip socket

Usage Guidelines

Use this command to collect information on the socket interface. To get more complete information on a socket/TCP port pair, use this command in conjunction with **debug ip tcp transactions**.

Because the socket debugging information is state-change oriented, you will not see the debug message on a per packet basis. However, if the connections normally have very short lives (few packet exchanges during the life cycle of a connection), then socket debugging could become expensive because of the state changes involved during connection setup and teardown.

Sample Displays

The following is sample output from the **debug ip socket** output from a server process:

```
Router# debug ip socket

Added socket 0x60B86228 to process 40
SOCKET: set TCP property TCP_PID, socket 0x60B86228, TCB 0x60B85E38
Accepted new socket fd 1, TCB 0x60B85E38
Added socket 0x60B86798 to process 40
SOCKET: set TCP property TCP_PID, socket 0x60B86798, TCB 0x60B877C0
SOCKET: set TCP property TCP_BIT_NOTIFY, socket 0x60B86798, TCB 0x60B877C0
SOCKET: created new socket to TCP, fd 2, TCB 0x60B877C0
SOCKET: bound socket fd 2 to TCB 0x60B877C0
SOCKET: set TCP property TCP_WINDOW_SIZE, socket 0x60B86798, TCB 0x60B877C0
SOCKET: listen on socket fd 2, TCB 0x60B877C0
SOCKET: closing socket 0x60B86228, TCB 0x60B85E38
SOCKET: socket event process: socket 0x60B86228, TCB new state --> FINWAIT1
socket state: SS_ISCONNECTED SS_CANTSENDMORE SS_ISDISCONNECTING
SOCKET: Removed socket 0x60B86228 from process 40 socket list
```

The following is sample output from the **debug ip socket** command from a client process:

```
Router# debug ip socket

Added socket 0x60B70220 to process 2
SOCKET: set TCP property TCP_PID, socket 0x60B70220, TCB 0x60B6CFDC
SOCKET: set TCP property TCP_BIT_NOTIFY, socket 0x60B70220, TCB 0x60B6CFDC
SOCKET: created new socket to TCP, fd 0, TCB 0x60B6CFDC
SOCKET: socket event process: socket 0x60B70220, TCB new state --> SYNSENT
socket state: SS_ISCONNECTING
SOCKET: socket event process: socket 0x60B70220, TCB new state --> ESTAB
socket state: SS_ISCONNECTING
SOCKET: closing socket 0x60B70220, TCB 0x60B6CFDC
SOCKET: socket event process: socket 0x60B70220, TCB new state --> FINWAIT1
socket state: SS_ISCONNECTED SS_CANTSENDMORE SS_ISDISCONNECTING
SOCKET: Removed socket 0x60B70220 from process 2 socket list
```

Table 71 describes the significant fields in the output.

Table 71 **Debug IP Socket Field Descriptions**

Field	Description
Added socket 0x60B86228 process 40	New socket is opened for process 40.
SOCKET	Indicates that this is a SOCKET transaction.
set TCP property TCP_PID	Set process ID to the TCP associated with the socket.
socket 0x60B86228, TCB 0x60B85E38	Address for the socket/TCP pair.
set TCP property TCP_BIT_NOTIFY	Set the method for how the socket wants to be notified for an event.
created new socket to TCP, fd 2	Opened a new socket referenced by file descriptor 2 to TCP.
bound socket fd 2 to TCB	Bound the socket referenced by file descriptor 2 to TCP.
listen on socket fd 2	Indicates which file descriptor the application is listening to.
closing socket	Indicates the socket is being closed.
socket event process	Processed a state change event occurred in the transport layer.
TCB new state --> FINWAIT1	TCP state machine changed to FINWAIT1. (See the debug ip tcp transaction command for more information on TCP state machines.)
socket state: SS_ISCONNECTED SS_CANTSENDMORE SS_ISDISCONNECTING	<p>New SOCKET state flags after the transport event processing. This socket is still connected, but disconnecting is in progress, and it will not send more data to peer.</p> <p>Possible SOCKET state flags follow:</p> <p>SS_NOFDREF No file descriptor reference for this socket.</p> <p>SS_ISCONNECTING Socket connecting is in progress.</p> <p>SS_ISBOUND Socket is bound to TCP.</p> <p>SS_ISCONNECTED Socket is connected to peer.</p> <p>SS_ISDISCONNECTING Socket disconnecting is in progress.</p> <p>SS_CANTSENDMORE Can't send more data to peer.</p> <p>SS_CANTRCVMORE Can't receive more data from peer.</p> <p>SS_ISDISCONNECTED Socket is disconnected. Connection is fully closed.</p>
Removed socket 0x60B86228 from process 40 socket list	Connection is closed, and the socket is removed from the process socket list.

Related Command**debug ip tcp transactions**

debug ip tcp driver

Use the **debug ip tcp driver** EXEC command to display information on Transmission Control Protocol (TCP) driver events; for example, connections opening or closing, or packets being dropped because of full queues. The **no** form of this command disables debugging output.

[no] debug ip tcp driver

Usage Guidelines

The TCP driver is the process that the router software uses to send packet data over a TCP connection. Remote source-route bridging, STUN (serial tunneling), and X.25 switching currently use the TCP driver.

Using the **debug ip tcp driver** command together with the **debug ip tcp driver-pak** command provides the most verbose debugging output concerning TCP driver activity.

Sample Display

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

```
Router# debug ip tcp driver

TCPDRV359CD8: Active open 172.21.80.26:0 --> 172.21.80.25:1996 OK, lport 36628
TCPDRV359CD8: enable tcp timeouts
TCPDRV359CD8: 172.21.80.26:36628 --> 172.21.80.25:1996 Abort
TCPDRV359CD8: 172.21.80.26:36628 --> 172.21.80.25:1996 DoClose tcp abort
```

Table 72 describes the fields in the first line of output.

Table 72 Debug IP TCP Driver Field Descriptions

Field	Description
TCPDRV359CD8:	Unique identifier for this instance of TCP driver activity.
Active open 172.21.80.26	Indication that the router at IP address 172.21.80.26 has initiated a connection to another router.
:0	The TCP port number the initiator of the connection uses to indicate that any port number can be used to set up a connection.
--> 172.21.80.25	The IP address of the remote router to which the connection has been initiated.
:1996	The TCP port number that the initiator of the connection is requesting that the remote router use for the connection. (1996 is a private TCP port number reserved in this implementation for remote source-route bridging.)
OK,	Indication that the connection has been established. If the connection has not been established, this field and the following field do not appear in this line of output.
lport 36628	The TCP port number that has actually been assigned for the initiator to use for this connection.

The following line indicates that the TCP driver user (remote source-route bridging, in this case) will allow TCP to drop the connection if excessive retransmissions occur:

```
TCPDRV359CD8: enable tcp timeouts
```

The following line indicates that the TCP driver user (in this case, remote source-route bridging) at IP address 172.21.80.26 (and using TCP port number 36628) is requesting that the connection to IP address 172.21.80.25 using TCP port number 1996 be aborted:

```
TCPDRV359CD8: 172.21.80.26:36628 --> 172.21.80.25:1996 Abort
```

The following line indicates that this connection was in fact closed due to an abnormal termination:

```
TCPDRV359CD8: 172.21.80.26:36628 --> 172.21.80.25:1996 DoClose tcp abort
```

debug ip tcp driver-pak

Use the **debug ip tcp driver-pak** EXEC command to display information on every operation that the Transmission Control Protocol (TCP) driver performs. The **no** form of this command disables debugging output.

[no] debug ip tcp driver-pak

Usage Guidelines

This command turns on a verbose debugging by logging at least one debugging message for every packet sent or received on the TCP driver connection.

The TCP driver is the process that the router software uses to send packet data over a TCP connection. Remote source-route bridging, STUN (serial tunneling), and X.25 switching currently use the TCP driver.

To observe the context within which certain **debug ip tcp driver-pak** messages occur, turn on this command in conjunction with the **debug ip tcp driver** command.

Note Because the **debug ip tcp driver-pak** command generates so many messages, use it only on lightly loaded systems. This command not only places a significant load on the system processor, but it may even change the symptoms of any unexpected behavior that occur.

Sample Display

The following is sample output from the **debug ip tcp driver-pak** command:

```
Router# debug ip tcp driver-pak

TCPDRV359CD8: send 2E8CD8 (len 26) queued
TCPDRV359CD8: output pak 2E8CD8 (len 26) (26)
TCPDRV359CD8: readf 42 bytes (Thresh 16)
TCPDRV359CD8: readf 26 bytes (Thresh 16)
TCPDRV359CD8: readf 10 bytes (Thresh 10)
TCPDRV359CD8: send 327E40 (len 4502) queued
TCPDRV359CD8: output pak 327E40 (len 4502) (4502)
```

Table 73 describes the fields shown in the first line.

Table 73 Debug TCP Driver-Pak Field Descriptions

Field	Description
TCPDRV359CD8	Unique identifier for this instance of TCP driver activity.
send	Indication that this event involves the TCP driver sending data.
2E8CD8	Address in memory of the data the TCP driver is sending.
(len 26)	Length of the data (in bytes).
queued	Indication that the TCP driver user process (in this case, remote source-route bridging) has transferred the data to the TCP driver to send.

The following line indicates that the TCP driver has sent the data that it had received from the TCP driver user, as shown in the previous line of output. The last field in the line (26) indicates that the 26 bytes of data were sent out as a single unit.

```
TCPDRV359CD8: output pak 2E8CD8 (len 26) (26)
```

The following line indicates that the TCP driver has received 42 bytes of data from the remote IP address. The TCP driver user (in this case, remote source-route bridging) has established an input threshold of 16 bytes for this connection. (The input threshold instructs the TCP driver to transfer data to the TCP driver user only when at least 16 bytes are present.)

```
TCPDRV359CD8: readf 42 bytes (Thresh 16)
```

debug ip tcp intercept

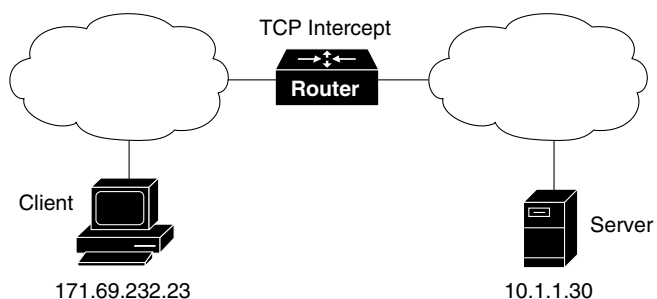
To display TCP intercept statistics, use the **debug ip tcp intercept EXEC** command. The **no** form of this command disables debugging output.

[no] debug ip tcp intercept

Sample Display

Figure 1 illustrates a scenario in which a router configured with TCP intercept operates between a client and a server.

Figure 1 Example TCP Intercept Environment



171.69.232.23 sends all packets
172.19.160.17 does not exist

55826

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

```
Router# debug ip tcp intercept
```

A connection attempt arrives:

```
INTERCEPT: new connection (172.19.160.17:61774) => (10.1.1.30:23)
INTERCEPT: 172.19.160.17:61774 <- ACK+SYN (10.1.1.30:61774)
```

A second connection attempt arrives:

```
INTERCEPT: new connection (172.19.160.17:62030) => (10.1.1.30:23)
INTERCEPT: 172.19.160.17:62030 <- ACK+SYN (10.1.1.30:62030)
```

The router retransmits to both apparent clients:

```
INTERCEPT: retransmit 2 (172.19.160.17:61774) <- (10.1.1.30:23) SYNRCVD
INTERCEPT: retransmit 2 (172.19.160.17:62030) <- (10.1.1.30:23) SYNRCVD
```

A third connection attempt arrives:

```
INTERCEPT: new connection (171.69.232.23:1048) => (10.1.1.30:23)
INTERCEPT: 171.69.232.23:1048 <- ACK+SYN (10.1.1.30:1048)
```

The router sends more retransmissions trying to establish connections with the apparent clients:

```
INTERCEPT: retransmit 4 (172.19.160.17:61774) <- (10.1.1.30:23) SYNRCVD
INTERCEPT: retransmit 4 (172.19.160.17:62030) <- (10.1.1.30:23) SYNRCVD
INTERCEPT: retransmit 2 (171.69.232.23:1048) <- (10.1.1.30:23) SYNRCVD
```

The router establishes the connection with the third client and retransmits to the server:

```
INTERCEPT: 1st half of connection is established (171.69.232.23:1048) => (10.1.1.30:23)
INTERCEPT: (171.69.232.23:1048) SYN -> 10.1.1.30:23
INTERCEPT: retransmit 2 (171.69.232.23:1048) -> (10.1.1.30:23) SYNSENT
```

The server responds; the connection is established:

```
INTERCEPT: 2nd half of connection established (171.69.232.23:1048) => (10.1.1.30:23)
INTERCEPT: (171.69.232.23:1048) ACK -> 10.1.1.30:23
```

The router retransmits to the first two apparent clients, times out, and sends resets:

```
INTERCEPT: retransmit 8 (172.19.160.17:61774) <- (10.1.1.30:23) SYNRCVD
INTERCEPT: retransmit 8 (172.19.160.17:62030) <- (10.1.1.30:23) SYNRCVD
INTERCEPT: retransmit 16 (172.19.160.17:61774) <- (10.1.1.30:23) SYNRCVD
INTERCEPT: retransmit 16 (172.19.160.17:62030) <- (10.1.1.30:23) SYNRCVD
INTERCEPT: retransmitting too long (172.19.160.17:61774) => (10.1.1.30:23) SYNRCVD
INTERCEPT: 172.19.160.17:61774 <- RST (10.1.1.30:23)
INTERCEPT: retransmitting too long (172.19.160.17:62030) => (10.1.1.30:23) SYNRCVD
INTERCEPT: 172.19.160.17:62030 <- RST (10.1.1.30:23)
```

debug ip tcp transactions

Use the **debug ip tcp transactions** EXEC command to display information on significant Transmission Control Protocol (TCP) transactions such as state changes, retransmissions, and duplicate packets. The **no** form of this command disables debugging output.

[no] debug ip tcp transactions

Usage Guidelines

This command is particularly useful for debugging a performance problem on a TCP/IP network that you have isolated above the data link layer.

The **debug ip tcp transactions** command displays output for packets the router sends and receives, but does not display output for packets it forwards.

Sample Display

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

```
Router# debug ip tcp transactions

TCP: sending SYN, seq 168108, ack 88655553
TCP0: Connection to 10.9.0.13:22530, advertising MSS 966
TCP0: state was LISTEN -> SYNRCVD [23 -> 10.9.0.13(22530)]
TCP0: state was SYNSENT -> SYNRCVD [23 -> 10.9.0.13(22530)]
TCP0: Connection to 10.9.0.13:22530, received MSS 956
TCP0: restart retransmission in 5996
TCP0: state was SYNRCVD -> ESTAB [23 -> 10.9.0.13(22530)]
TCP2: restart retransmission in 10689
TCP2: restart retransmission in 10641
TCP2: restart retransmission in 10633
TCP2: restart retransmission in 13384 -> 10.0.0.13(16151)]
TCP0: restart retransmission in 5996 [23 -> 10.0.0.13(16151)]
```

Table 74 describes significant fields in the output.

Table 74 Debug IP TCP Transactions Field Descriptions

Field	Description
TCP:	Indicates that this is a TCP transaction.
sending SYN	Indicates that a synchronize packet is being sent.
seq 168108	Indicates the sequence number of the data being sent.
ack 88655553	Indicates the sequence number of the data being acknowledged.
TCP0:	Indicates the TTY number (0, in this case) with which this TCP connection is associated.
Connection to 10.9.0.13:22530	Indicates the remote address with which a connection has been established.
advertising MSS 966	Indicates the maximum segment size this side of the TCP connection is offering to the other side.

Table 74 Debug IP TCP Transactions Field Descriptions (Continued)

Field	Description
state was LISTEN -> SYNRCVD	<p>Indicates that the TCP state machine changed state from LISTEN to SYNSENT. Possible TCP states follow:</p> <p>CLOSED—Connection closed.</p> <p>CLOSEWAIT—Received a FIN segment.</p> <p>CLOSING—Received a FIN/ACK segment.</p> <p>ESTAB—Connection established.</p> <p>FINWAIT 1—Sent a FIN segment to start closing the connection.</p> <p>FINWAIT 2—Waiting for a FIN segment.</p> <p>LASTACK—Sent a FIN segment in response to a received FIN segment.</p> <p>LISTEN—Listening for a connection request.</p> <p>SYNRCVD—Received a SYN segment, and responded.</p> <p>SYNSENT—Sent a SYN segment to start connection negotiation.</p> <p>TIMEWAIT—Waiting for network to clear segments for this connection before the network no longer recognizes the connection as valid. This must occur before a new connection can be set up.</p>
[23 -> 10.9.0.13(22530)]	<p>Within these brackets:</p> <p>The first field (23) indicates local TCP port.</p> <p>The second field (10.9.0.13) indicates the destination IP address.</p> <p>The third field (22530) indicates the destination TCP port.</p>
restart retransmission in 5996	<p>Indicates the number of milliseconds until the next retransmission takes place.</p>

debug ip trigger-authentication

Use the **debug ip trigger-authentication** command to display information related to automated double authentication. The **no** form of this command disables debugging output.

[no] debug ip trigger-authentication [verbose]

Syntax Description

verbose (Optional) Specifies that the complete debugging output be displayed, including information about packets that are blocked before authentication is complete.

Usage Guidelines

Use this command when troubleshooting automated double authentication.

This command displays information about the remote host table—whenever entries are added, updated, or removed, a new debugging message is displayed.

What is the remote host table? Whenever a remote user needs to be user-authenticated in the second stage of automated double authentication, the local device sends a UDP packet to the remote user's host. Whenever such a UDP packet is sent, the user's host IP address is added to a table. If additional UDP packets are sent to the same remote host, a new table entry is not created; instead, the existing entry is updated with a new time stamp. This remote host table contains a cumulative list of host entries; entries are deleted after a timeout period or after you manually clear the table using the **clear ip trigger-authentication** command.

If you include the **verbose** keyword, the debugging output also includes information about packet activity.

Sample Display

The following is sample output from the **debug ip trigger-authentication** command. In this example, the local device at 172.21.127.186 sends a UDP packet to the remote host at 172.21.127.114. The UDP packet is sent to request the remote user's username and password (or PIN). (The output indicates "New entry added.")

After a timeout period, the local device has not received a valid response from the remote host, so the local device sends another UDP packet. (The output indicates "Time stamp updated.")

Then the remote user is authenticated, and after a length of time (the timeout period) the entry is removed from the remote host table. (The output indicates "remove obsolete entry.")

```
myfirewall# debug ip trigger-authentication
TRIGGER_AUTH: UDP sent from 172.21.127.186 to 172.21.127.114, qdata=7C2504
                New entry added, timestamp=2940514234
TRIGGER_AUTH: UDP sent from 172.21.127.186 to 172.21.127.114, qdata=7C2504
                Time stamp updated, timestamp=2940514307
TRIGGER_AUTH: remove obsolete entry, remote host=172.21.127.114
```

The following is sample output from the **debug ip trigger-authentication verbose** command. In this example, messages about packet activity are included because of the use of the **verbose** keyword.

You can see many packets that are being blocked at the interface because the user has not yet been double authenticated. These packets will be permitted through the interface only after the user has been double authenticated. (You can see packets being blocked when the output indicates “packet enqueued” then “packet ignored.”)

```
TRIGGER_AUTH: packet enqueued, qdata=69FEEC
                remote host=172.21.127.113, local host=172.21.127.186 (if: 0.0.0.0)
TRIGGER_AUTH: UDP sent from 172.21.127.186 to 172.21.127.113, qdata=69FEEC
                Time stamp updated
TRIGGER_AUTH: packet enqueued, qdata=69FEEC
                remote host=172.21.127.113, local host=172.21.127.186 (if: 0.0.0.0)
TRIGGER_AUTH: packet ignored, qdata=69FEEC
TRIGGER_AUTH: packet enqueued, qdata=69FEEC
                remote host=172.21.127.113, local host=172.21.127.186 (if: 0.0.0.0)
TRIGGER_AUTH: packet ignored, qdata=69FEEC
TRIGGER_AUTH: packet enqueued, qdata=69FEEC
                remote host=172.21.127.113, local host=172.21.127.186 (if: 0.0.0.0)
TRIGGER_AUTH: UDP sent from 172.21.127.186 to 172.21.127.113, qdata=69FEEC
                Time stamp updated
TRIGGER_AUTH: packet enqueued, qdata=69FEEC
                remote host=172.21.127.113, local host=172.21.127.186 (if: 0.0.0.0)
TRIGGER_AUTH: packet ignored, qdata=69FEEC
TRIGGER_AUTH: packet enqueued, qdata=69FEEC
                remote host=172.21.127.113, local host=172.21.127.186 (if: 0.0.0.0)
TRIGGER_AUTH: packet ignored, qdata=69FEEC
```

debug ip udp

To enable logging of User Datagram Protocol (UDP) packets sent and received, use the **debug ip udp** privileged EXEC command. To disable debugging output, use the **no** form of this command.

[no] debug ip udp

Usage Guidelines

Enter the **debug ip udp** command on the device that should be receiving packets from the host. Check the debugging output to see whether packets are being received from the host.



Caution The **debug ip udp** command can use considerable CPU cycles on the device. Do not enable it if your network is heavily congested

Sample Display

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

```
Router# debug ip udp
UDP packet debugging is on
Router#

00:18:48: UDP: rcvd src=0.0.0.0(68), dst=255.255.255.255(67), length=584
00:18:48: UDP: sent src=10.1.1.10(67), dst=172.17.110.136(67), length=604
00:18:48: UDP: rcvd src=172.17.110.136(67), dst=10.1.1.10(67), length=308
00:18:48: UDP: sent src=0.0.0.0(67), dst=255.255.255.255(68), length=328
00:18:48: UDP: rcvd src=0.0.0.0(68), dst=255.255.255.255(67), length=584
00:18:48: UDP: sent src=10.1.1.10(67), dst=172.17.110.136(67), length=604
00:18:48: UDP: rcvd src=172.17.110.136(67), dst=10.1.1.10(67), length=308
00:18:50: UDP: sent src=0.0.0.0(67), dst=255.255.255.255(68), length=328
```

debug ip wccp events

Use the **debug ip wccp events** EXEC command to display information about significant Web Cache Control Protocol events. The **no** form of this command disables debugging output.

[no] debug ip wccp events

Sample Display

The following is sample output from the **debug ip wccp events** command when a Cisco Cache Engine is added to the list of available Web caches.

```
Router# debug ip wccp events

WCCP-EVNT: Built I_See_You msg body w/1 usable web caches, change # 0000000A
WCCP-EVNT: Web Cache 192.168.25.3 added
WCCP-EVNT: Built I_See_You msg body w/2 usable web caches, change # 0000000B
WCCP-EVNT: Built I_See_You msg body w/2 usable web caches, change # 0000000C
```

debug ip wccp packets

Use the **debug ip wccp packets** EXEC command to display information about every Web Cache Control Protocol packet received or sent by the router. The **no** form of this command disables debugging output.

[no] debug ip wccp packets

Sample Display

The following is sample output from the **debug ip wccp packets** command. The router is sending keepalive packets to the Cisco Cache Engines at 192.168.25.4 and 192.168.25.3. Each keepalive packet has an identification number associated with it. When the Cisco Cache Engine receives a keepalive packet from the router, it sends a reply with the identification number back to the router.

```
Router# debug ip wccp packets

WCCP-PKT: Received valid Here_I_Am packet from 192.168.25.4 w/rcvd_id 00003532
WCCP-PKT: Sending I_See_You packet to 192.168.25.4 w/ rcvd_id 00003534
WCCP-PKT: Received valid Here_I_Am packet from 192.168.25.3 w/rcvd_id 00003533
WCCP-PKT: Sending I_See_You packet to 192.168.25.3 w/ rcvd_id 00003535
WCCP-PKT: Received valid Here_I_Am packet from 192.168.25.4 w/rcvd_id 00003534
WCCP-PKT: Sending I_See_You packet to 192.168.25.4 w/ rcvd_id 00003536
WCCP-PKT: Received valid Here_I_Am packet from 192.168.25.3 w/rcvd_id 00003535
WCCP-PKT: Sending I_See_You packet to 192.168.25.3 w/ rcvd_id 00003537
WCCP-PKT: Received valid Here_I_Am packet from 192.168.25.4 w/rcvd_id 00003536
WCCP-PKT: Sending I_See_You packet to 192.168.25.4 w/ rcvd_id 00003538
WCCP-PKT: Received valid Here_I_Am packet from 192.168.25.3 w/rcvd_id 00003537
WCCP-PKT: Sending I_See_You packet to 192.168.25.3 w/ rcvd_id 00003539
```