Table of Contents

Important Information on Debug Commands

Introduction

This page provides some general guidelines on using the debugs available on Cisco IOS® platforms, as well as examples for properly using the **debug ip packet** command and conditional debugging.

Note: This document does not explain how to use and interpret specific **debug** commands and outputs. Refer to the appropriate Cisco Debug Command Reference documentation for information on specific **debug** commands.

The output from **debug** privileged EXEC commands provides diagnostic information concerning a variety of internetworking events relating to protocol status and network activity in general.

Prerequisites

Requirements

Readers of this document should be knowledgeable of the following:

- Connecting to the router using the console, aux and vty ports.
- General IOS configuration issues.
- Interpreting IOS debug outputs

Components Used

This document is not restricted to specific software and hardware versions.

The information presented in this document was created from devices in a specific lab environment. All of the devices used in this document started with a cleared (default) configuration. If you are working in a live network, ensure that you understand the potential impact of any command before using it.

Warnings

Use **debug** commands with caution. In general, it is recommended that these commands only be used under

the direction of your router technical support representative when troubleshooting specific problems.

Enabling debugging can disrupt operation of the router when internetworks are experiencing high load conditions. Hence, if logging is enabled, the access server may intermittently freeze up as soon as the console port gets overloaded with log messages.

Before you start a **debug** command, always consider the output that this command will generate and the amount of time this may take. For example, if you have a router with one basic rate interface (BRI), **debug isdn q931** probably won't harm the system. However, doing the same debug on an AS5800 with full E1 configuration will probably generate so much input that it may "hang" and stop responding.

Before debugging, look at your CPU load using the **show processes cpu** command. Verify that you have ample CPU available before beginning the debugs. For more information on handling high CPU loads, refer to the document Troubleshooting High CPU Utilization on Cisco Routers. For example, if you have a Cisco 7200 router with an ATM interface doing bridging then –– depending on the amount of subinterfaces configured –– restarting the router might use a lot of its CPU. The reason here is that, for each virtual circuit (VC), a Bridge Protocol Data Unit (BPDU) packet needs to be generated. Starting debugs during such a critical time may cause the CPU utilization to rise dramatically and result in a hang or network connectivity loss.

Note: When debugs are running, you don't usually see the router prompt, especially when the debug is intensive. However, in most cases, you can use the **no debug all** or **undebug all** commands to stop the debugs. Refer to the section Obtaining Debug Outputs for more information on safely using debugs

Conventions

For more information on document conventions, see the Cisco Technical Tips Conventions.

Before Debugging

In addition to the points mentioned above, make sure you understand the impact of the debugs on the stability of the platform. You should also consider which interface on the router you should connect to. The following section has some guidelines.

Obtaining Debug Outputs

Routers can display debug outputs to various interfaces, including the console, aux, and vty ports. Routers can also log messages to an internal buffer to an external unix syslog server. Instructions and caveats for each method are discussed below:

Console Port

If you are connected on the console, under normal configurations, no extra works needs to be done. The debug output should be automatically displayed. However, make sure the **logging console level** is set as desired and that logging has not been disabled with the **no logging console** command. See the document Using Debug Commands for more information.

 Warning: Excessive debugs to the console port of a router can cause it to hang. This is because the

router automatically prioritizes console output ahead of other router functions. Hence if the router is processing a large debug output to the console port, it may hang. Hence, if the debug output is excessive use

the vty (telnet) ports or the log buffers to obtain your debugs. More information is provided below.

Note: By default, logging is enabled on the console port. Hence, the console port always processes debug output even if you are actually using some other port or method (such as Aux, vty or buffer) to capture the output. Hence, we recommend that, under normal operating conditions, you have the command **no logging console** enabled at all times and use other methods to capture debugs. In situations where you need to use the console, temporarily turn **logging console** back on.

Aux Port

If you are connected via an Auxiliary port, type the command **terminal monitor**. Also verify that the **no logging on** command has not been activated on the router.

Note: If you use the Aux port to monitor the router, keep in mind that, when the router reboots, the Aux port does not display the boot sequence output. To view the boot sequence, connect to the console port.

VTY Ports

If you are connected via an Auxiliary port or via telnet, type the command **terminal monitor**. Also verify that the **no logging on** command has not been used.

Logging Messages to an Internal Buffer

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

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

```
logging buffered
no logging buffered
```

The **logging buffered** command copies log messages to an internal buffer instead of writing them to the console. The buffer is circular in nature, so newer messages overwrite older messages. To display the messages that are logged in the buffer, use the privileged EXEC command **show logging**. The first message displayed is the oldest message in the buffer. You can specify the size of the buffer as well as the severity level of the messages to be logged.

Tip: Make sure enough memory is available in the box before entering the buffer size. Use the IOS command **show proc mem** to see memory available.

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

Logging Messages to a UNIX Syslog Server

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

```
logging <ip-address>
no logging <ip-address>
```

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

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

For more information on setting up a syslog server refer to the document Using Debug Commands.

Other Pre–Debug Tasks

1. Setup your terminal emulator software (for example, HyperTerminal) so that it can capture the debug output to a file. For example, in HyperTerminal, click **Transfer**, then click **Capture Text**, and choose the appropriate options. For more information, refer to the document Capturing Text Output from Hyperterminal. For other terminal emulator software, refer to the software documentation.
2. Enable millisecond (msec) timestamps using the **service timestamps** command:

```
router(config)#service timestamps debug datetime msec
        router(config)#service timestamps log datetime msec
```

These commands add time stamps to debugs in the format MMM DD HH:MM:SS, indicating the date and time according to the system clock. If the system clock has not been set, the date and time are preceded by an asterisk (*) to indicate that the date and time are probably not correct.

It is generally advisable to configure millisecond timestamps as this provides high level of clarity when looking at debug outputs. Millisecond timestamps provide a better indication of the timing of the various debugs events relative to each other. However, note that, when the console port outputs a lot of messages, they might not correlate with the actual timing of the event. For example, if we enable **debug x25** all on a box that has 200 VCs, and the output is logged to the buffer (using the **no logging console** and **logging buffered** commands), the timestamp displayed in the debug output (within the buffer) might not be the exact time when the packet passes through the interface. Therefore, do not use msec timestamps to prove performance issues, but to obtain relative information on when events occur.

To Stop Debugging

To stop a debug, use the **no debug all** or **undebug all** commands. Verify that the debugs have been turned off using the command **show debug**.

Remember that the commands **no logging console** and **terminal no monitor** only prevent the output from being output on the console, Aux or vty respectively. It does not stop the debugging and therefore uses up router resources.

Using the debug ip packet Command

The **debug ip packet** command produces information on packets that are not fast switched by the router. However, since it generates an output for every packet, the output can be extensive and thus cause the router to hang. For this reason, only use **debug ip packet** under the strictest controls as described in this section.

The best way to limit the output of **debug ip packet** is to create an access–list that linked to the debug. Only packets that match the access–list criteria will be subject to **debug ip packet**. This access–list does not need to be applied on any interface, but rather is applied to the debug operation.

Before using **debugging ip packet**, note that the router is doing fast−switching by default, or may be doing CEF switching if configured to do so. This means that, once those techniques are in place, the packet is not provided to the processor, hence the debugging does not show anything. For this to work, you need to disable fast−switching on the router with **no ip route−cache** (for unicast packets) or **no ip mroute−cache** (for multicast packets). This should be applied on the interfaces where the traffic is supposed to flow. Verify this with the **show ip route** command.

Warnings:

- Disabling fast−switching on a router that handles a large number of packets can cause CPU utilization to spike so that the box hangs or loses its connection to its peers.
- Do not disable fast−switching on a router running Multi Protocol Label Switching (MPLS). MPLS is used in conjunction with CEF. Therefore, disabling fast−switching on the interface can have disastrous effect.

Let us consider a sample scenario:

router_121   router_122   router_123

The access−list configured on router_122 is :

```
access-list 105 permit icmp host 10.10.10.2 host 13.1.1.1
access-list 105 permit icmp host 13.1.1.1 host 10.10.10.2
```

This access list permits any Internet Control Message Protocol (ICMP) packet from host router_121 (with IP address 10.10.10.2) to host router_123 (with IP address 13.1.1.1) as well as in the other direction. It is important that you permit the packets in either direction, else the router may drop the returning ICMP packet.

Now let's remove fast−switching on only one interface on router_122. This means that we can only see the debugs for the packets that are destined for that interface, as seen from the perspective of the IOS intercepting the packet. From the debugs, such packets will appear with "d=". Since we have not yet turned off fast switching on the other interface, the return packet will not be subject to **debug ip packet**. The following output shows how we disable fast switching:

```
router_122(config)#interface virtual-template 1
router_122(config-if)#no ip route-cache
router_122(config-if)#end
```

We must now activate **debug ip packet** with the access−list defined earlier (access−list 105).

```
router_122#debug ip packet detail 105
IP packet debugging is on (detailed) for access list 105
router_122#
00:10:01: IP: s=13.1.1.1 (Serial3/0), d=10.10.10.2 (Virtual-Access1),
g=10.10.10.2, len 100, forward

00:10:01:     ICMP type=0, code=0

! -- ICMP packet from 13.1.1.1 to 10.10.10.2.
! -- This packet is displayed because it matches the
```

Cisco – Important Information on Debug Commands

```
                      ! -- source and destination requirements in access list 105

             00:10:01: IP: s=13.1.1.1 (Serial3/0), d=10.10.10.2 (Virtual-Access1),
             g=10.10.10.2, len 100, forward
             00:10:01:     ICMP type=0, code=0
             00:10:01: IP: s=13.1.1.1 (Serial3/0), d=10.10.10.2 (Virtual-Access1),
             g=10.10.10.2, len 100, forward
             00:10:01:     ICMP type=0, code=0
```

Now let's remove fast–switching on the other interface (on router_122). This means that all packets across those two interfaces are now packet–switched (which is a requirement for **debug ip packet**):

```
             router_122(config)#interface serial 3/0
             router_122(config-if)#no ip route-cache
             router_122(config-if)#end

             router_122#
             00:11:57:  IP: s=10.10.10.2 (Virtual-Access1), d=13.1.1.1
             (Serial3/0), g=172.16.1.6, len 100, forward
             00:11:57:  ICMP type=8, code=0

                      ! -- ICMP packet (echo) from 10.10.10.2 to 13.1.1.1

             00:11:57: IP: s=13.1.1.1 (Serial3/0), d=10.10.10.2 (Virtual-Access1),
             g=10.10.10.2, len 100, forward
             00:11:57:  ICMP type=0, code=0

                      ! -- ICMP return packet (echo-reply) from 13.1.1.1 to 10.10.10.2

             00:11:57:  IP: s=10.10.10.2 (Virtual-Access1), d=13.1.1.1 (Serial3/0),
             g=172.16.1.6, len 100, forward
             00:11:57:  ICMP type=8, code=0
             00:11:57:  IP: s=13.1.1.1 (Serial3/0), d=10.10.10.2 (Virtual-Access1),
             g=10.10.10.2, len 100, forward
             00:11:57:  ICMP type=0, code=0
```

Note that the debug ip packet output does not show any packets that do not match the access–list criteria. For some additional information on this procedure, refer to the document Understanding the Ping and Traceroute Commands.

For more information on how to build access–lists, refer to the document Configuring IP Access Lists.

Conditionally Triggered Debugs

When the Conditionally Triggered Debugging feature is enabled, the router generates debugging messages for packets entering or leaving the router on a specified interface; the router does not generate debugging output for packets entering or leaving through a different interface. For information regarding the uses for Conditional Debugs refer to the document Conditionally Triggered Debugging.

Let's look at a simple implementation of conditional debugs. Consider the following scenario: the router shown below (trabol) has two interfaces (serial 0 and serial 3) both running HDLC encapsulation

We use the normal **debug serial interface** command to observe the HDLC keepalives received on all interfaces. We can observe the keepalives on both interfaces.

```
             traxbol#debug serial interface
             Serial network interface debugging is on
             traxbol#
```

Cisco – Important Information on Debug Commands

```
    *Mar  8 09:42:34.851: Serial0: HDLC myseq 28, mineseen 28*, yourseen 41, line up

    ! -- HDLC keeplaive on interface Serial 0

    *Mar  8 09:42:34.855: Serial3: HDLC myseq 26, mineseen 26*, yourseen 27, line up

    ! -- HDLC keeplaive on interface Serial 3

    *Mar  8 09:42:44.851: Serial0: HDLC myseq 29, mineseen 29*, yourseen 42, line up
    *Mar  8 09:42:44.855: Serial3: HDLC myseq 27, mineseen 27*, yourseen 28, line up
```

Now let's enable conditional debugs for interface serial 3. This means that only debugs for interface serial 3 are displayed. Use the command **debug interface** *<interface_type interface_number>*.

```
    traxbol#debug interface serial 3
    Condition 1 set
```

Use the **show debug condition** command to verify that the condional debug is active. Note that a condition for interface serial 3 is active.

```
    traxbol#show debug condition

    Condition 1: interface Se3 (1 flags triggered)
    Flags: Se3
    traxbol#
```

Note that now only the debugs for interface serial 3 are displayed

```
    *Mar  8 09:43:04.855: Serial3: HDLC myseq 29, mineseen 29*, yourseen 30, line up
    *Mar  8 09:43:14.855: Serial3: HDLC myseq 30, mineseen 30*, yourseen 31, line up
```

To remove the conditional debug use the command **undebug interface** *<interface_type interface_number>*. It is recommended that you turn off the debugs (for example, using undebug all) before removing the conditional trigger. This is to avoid a deluge of debug outputs when the condition is removed.

```
    traxbol#undebug interface serial 3
    This condition is the last interface condition set.
    Removing all conditions may cause a flood of debugging
    messages to result, unless specific debugging flags
    are first removed.
    Proceed with removal? [yes/no]: y
    Condition 1 has been removed
    traxbol#
```

We now observe that debug for both interface serial 0 as well as serial 3 are displayed.

```
    *Mar  8 09:43:34.927: Serial3: HDLC myseq 32, mineseen 32*, yourseen 33, line up
    *Mar  8 09:43:44.923: Serial0: HDLC myseq 35, mineseen 35*, yourseen 48, line up
```

⚠️ **Warning:** Some debugging operations are conditional by themselves. An example is atm debugging.

With ATM debugging you should explicitly specify the interface for which debugs should be enabled rather than enabling debugs on all atm interfaces and specifying a condition.

The following section shows the correct way to limit ATM packet debugging to one subinterface:

```
    arielle-nrp2#debug atm packet interface atm 0/0/0.1
```

```
                !--- Note that we explicitly specify the sub-interface to be used for debugging

            ATM packets debugging is on
            Displaying packets on interface ATM0/0/0.1 only
            arielle-nrp2#
            *Dec 21 10:16:51.891: ATM0/0/0.1(O):
            VCD:0x1 VPI:0x1 VCI:0x21 DM:0x100 SAP:AAAA CTL:03 OUI:0080C2 TYPE:0007
            Length:0x278
            *Dec 21 10:16:51.891: 0000 FFFF FFFF FFFF 0010 7BB9 BDC4 0800 4500 025C 01FE
            0000 FF11 61C8 0A30
            *Dec 21 10:16:51.891: 4B9B FFFF FFFF 0044 0043 0248 0000 0101 0600 0015 23B7
            0000 8000 0000 0000
            *Dec 21 10:16:51.891: 0000 0000 0000 0000 0000 0000 0010 7BB9 BDC3 0000 0000
            0000 0000 0000 0000
            *Dec 21 10:16:51.891: 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
            0000 0000 0000 0000
            *Dec 21 10:16:51.891: 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
            0000 0000 0000 0000
            *Dec 21 10:16:51.891: 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
            0000 0000 0000 0000
            *Dec 21 10:16:51.891: 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
            0000 0000 0000 0000
            *Dec 21 10:16:51.895: 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
            0000 0000 0000 0000
            *Dec 21 10:16:51.895:
            arielle-nrp2#
```

If you try to enable **atm debugging** on all interfaces (with an applied condition), the router may hang if it has a large number of ATM sub–interfaces. An example of the incorrect method for atm debugging is shown.

In this case you can see that a condition is applied but you also see that this has no effect. We can still see the packet from the other interface. In this lab scenario we have only two interfaces and very little traffic. If the number of interfaces is high, then the debug output for all the interfaces will be extremely high and it could cause the router to hang.

```
            arielle-nrp2#show debugging condition
            Condition 1: interface AT0/0/0.1 (1 flags triggered)
            Flags: AT0/0/0.1

            ! -- A condition for a specific interface.


            arielle-nrp2#debug atm packet
            ATM packets debugging is on
            Displaying all ATM packets
            arielle-nrp2#
            *Dec 21 10:22:06.727: ATM0/0/0.2(O):

            ! -- We see debugs from interface ATM0/0/0/.2, even though the condition
            ! -- specified ONLY AT0/0/0.1


              VCD:0x2 VPI:0x5 VCI:0x37 DM:0x100 SAP:AAAA CTL:03 OUI:0080C2
            TYPE:000E Length:0x2F
            *Dec 21 10:22:06.727: 0000 0000 0180 0000 107B B9BD C400 0000 0080
            0000 107B B9BD C480 0800 0014
            *Dec 21 10:22:06.727: 0002 000F 0000
            *Dec 21 10:22:06.727: un a
            *Dec 21 10:22:08.727: ATM0/0/0.2(O):
              VCD:0x2 VPI:0x5 VCI:0x37 DM:0x100 SAP:AAAA CTL:03 OUI:0080C2
            TYPE:000E Length:0x2F
            *Dec 21 10:22:08.727: 0000 0000 0180 0000 107B B9BD C400 0000 0080
```

```
                0000 107B B9BD C480 0800 0014
*Dec 21 10:22:08.727: 0002 000F 0000
*Dec 21 10:22:08.727: ll
*Dec 21 10:22:10.727: ATM0/0/0.2(O):
  VCD:0x2 VPI:0x5 VCI:0x37 DM:0x100 SAP:AAAA CTL:03 OUI:0080C2
TYPE:000E Length:0x2F
*Dec 21 10:22:10.727: 0000 0000 0080 0000 107B B9BD C400 0000 0080
0000 107B B9BD C480 0800 0014
*Dec 21 10:22:10.727: 0002 000F 0000
*Dec 21 10:22:10.727:
*Dec 21 10:22:12.727: ATM0/0/0.2(O):
  VCD:0x2 VPI:0x5 VCI:0x37 DM:0x100 SAP:AAAA CTL:03 OUI:0080C2
TYPE:000E Length:0x2F
*Dec 21 10:22:12.727: 0000 0000 0080 0000 107B B9BD C400 0000 0080
0000 107B B9BD C480 0800 0014
*Dec 21 10:22:12.727: 0002 000F 0000
*Dec 21 10:22:12.727:
*Dec 21 10:22:13.931: **ATM0/0/0.1**(O):
```

!--- We also see debugs for interface ATM0/0/0.1 as we wanted.

```
VCD:0x1 VPI:0x1 VCI:0x21 DM:0x100 SAP:AAAA CTL:03 OUI:0080C2
TYPE:0007 Length:0x278
*Dec 21 10:22:13.931: 0000 FFFF FFFF FFFF 0010 7BB9 BDC4 0800 4500
025C 027F 0000 FF11 6147 0A30
*Dec 21 10:22:13.931: 4B9B FFFF FFFF 0044 0043 0248 0000 0101 0600
001A 4481 0000 8000 0000 0000
*Dec 21 10:22:13.931: 0000 0000 0000 0000 0000 0000 0010 7BB9 BDC3
0000 0000 0000 0000 0000 0000
*Dec 21 10:22:13.931: 0000 0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000
*Dec 21 10:22:13.931: 0000 0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000
*Dec 21 10:22:13.931: 0000 0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000
*Dec 21 10:22:13.931: 0000 0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000
*Dec 21 10:22:13.935: 0000 0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000
```

Related Information

- **Access Technology Support Pages**
- **Technical Support – Cisco Systems**