

## debug tarp events

Use the **debug tarp events** EXEC command to display information on Target Identifier Address Resolution Protocol (TARP) activity. The **no** form of this command disables debugging output.

**[no] debug tarp events**

### Usage Guidelines

For complete information on the TARP process, use the **debug tarp packets** command along with the **debug tarp events** command. Events are usually related to error conditions.

### Sample Display

The following is sample output from the **debug tarp events** and **debug tarp packets** commands after the **tarp resolve** command was used to determine the NSAP address for the TARP target identifier (TID) *artemis*.

```
Router# debug tarp events
Router# debug tarp packets
Router# tarp resolve artemis

Type escape sequence to abort.
Sending TARP type 1 PDU, timeout 15 seconds...

NET corresponding to TID artemis is 49.0001.1111.1111.1111.00

*Mar 1 00:43:59: TARP-PA: Propagated TARP packet, type 1, out on Ethernet0
*Mar 1 00:43:59:      Lft = 100, Seq = 11, Prot type = 0xFE, URC = TRUE
*Mar 1 00:43:59:      Ttid len = 7, Stid len = 8, Prot addr len = 10
*Mar 1 00:43:59:      Destination NSAP: 49.0001.1111.1111.1111.00
*Mar 1 00:43:59:      Originator's NSAP: 49.0001.3333.3333.3333.00
*Mar 1 00:43:59:      Target TID: artemis
*Mar 1 00:43:59:      Originator's TID: cerd
*Mar 1 00:43:59: TARP-EV: Packet not propagated to 49.0001.4444.4444.4444.00 on
      interface Ethernet0 (adjacency is not in UP state)
*Mar 1 00:43:59: TARP-EV: No route found for TARP static adjacency
      55.0001.0001.1111.1111.1111.1111.1111.1111.1111.00 - packet not sent
*Mar 1 00:43:59: TARP-PA: Received TARP type 3 PDU on interface Ethernet0
*Mar 1 00:43:59:      Lft = 100, Seq = 5, Prot type = 0xFE, URC = TRUE
*Mar 1 00:43:59:      Ttid len = 0, Stid len = 7, Prot addr len = 10
*Mar 1 00:43:59:      Packet sent/propagated by 49.0001.1111.1111.1111.af
*Mar 1 00:43:59:      Originator's NSAP: 49.0001.1111.1111.1111.00
*Mar 1 00:43:59:      Originator's TID: artemis
*Mar 1 00:43:59: TARP-PA: Created new DYNAMIC cache entry for artemis
```

Table 167 describes the significant fields in this display.

**Table 167** Debug TARP Field Descriptions—TARP Resolve Command

Field	Descriptions
Sending TARP type 1 PDU	PDU requesting the NSAP of the specified TID.
timeout	Number of seconds the router will wait for a response from the Type 1 PDU. The timeout is set by the <b>tarp t1-response-timer</b> command.
NET corresponding to	NSAP address (in this case, 49.0001.1111.1111.1111.00) for the specified TID.
*Mar 1 00:43:59	Debug timestamp.

**Table 167      Debug TARP Field Descriptions—TARP Resolve Command (Continued)**

<b>Field</b>	<b>Descriptions</b>
TARP-PA: Propagated	TARP packet: A Type 1 PDU was sent out on interface Ethernet 0.
Lft	Lifetime of the PDU (in hops).
Seq	Sequence number of the PDU.
Prot type	Protocol type of the PDU.
URC	The update remote cache bit.
Ttid len	Destination TID length.
Stid len	Source TID length.
Prot addr len	Protocol address length (bytes).
Destination NSAP	NSAP address that the PDU is being sent to.
Originator's NSAP	NSAP address that the PDU was sent from.
Target TID	TID that the PDU is being sent to.
Originator's TID	TID that the PDU was sent from.
TARP-EV: Packet not propagated	TARP event: The Type 1 PDU was not propagated on interface Ethernet 0 because the adjacency is not up.
TARP-EV: No route found	TARP event: The Type 1 PDU was not sent because no route was available.
TARP-PA: Received TARP Packet sent/propagated by	TARP packet: A Type 3 PDU was received on interface Ethernet 0. NSAP address of the router that sent or propagated the PDU.
TARP-PA: Created new DYNAMIC cache entry	TARP packet: A dynamic entry was made to the local TID cache.

Related Command

**debug tarp packets**

## debug tarp packets

Use the **debug tarp packets** EXEC command to display general information on TARP packets received, generated, and propagated on the router. The **no** form of this command disables debugging output.

**[no] debug tarp packets**

### Usage Guidelines

For complete information on the TARP process, use the **debug tarp events** command along with the **debug tarp packet** command. Events are usually related to error conditions.

### Sample Display

The following is sample output from the **debug tarp packet** command after the **tarp query** command was used to determine the TID for the NSAP address 49.0001.3333.3333.3333.00.

```
Router# debug tarp packets
Router# debug tarp events
Router# tarp query 49.0001.3333.3333.3333.00

Type escape sequence to abort.
Sending TARP type 5 PDU, timeout 40 seconds...

TID corresponding to NET 49.0001.3333.3333.3333.00 is cerdiwen

*Mar 2 03:10:11: TARP-PA: Originated TARP packet, type 5, to destination
49.0001.3333.3333.3333.00
*Mar 2 03:10:11: TARP-PA: Received TARP type 3 PDU on interface Ethernet0
*Mar 2 03:10:11:      Lft = 100, Seq = 2, Prot type = 0xFE, URC = TRUE
*Mar 2 03:10:11:      Ttid len = 0, Stid len = 8, Prot addr len = 10
*Mar 2 03:10:11:      Packet sent/propagated by 49.0001.3333.3333.3333.af
*Mar 2 03:10:11:      Originator's NSAP: 49.0001.3333.3333.3333.00
*Mar 2 03:10:11:      Originator's TID: cerdiwen
*Mar 2 03:10:11: TARP-PA: Created new DYNAMIC cache entry for cerdiwen
```

Table 168 describes the fields shown in the display.

**Table 168** Debug TARP Field Descriptions—TARP Query Command

Field	Descriptions
Sending TARP type 5 PDU	PDU requesting the TID of the specified NSAP.
timeout	Number of seconds the router will wait for a response from the Type 5 PDU. The timeout is set by the <b>tarp arp-request-timer</b> command.
TID corresponding to NET	TID (in this case <i>cerdiwen</i> ) for the specified NSAP address.
*Mar 2 03:10:11	Debug timestamp.
TARP-PA: Originated TARP packet	TARP packet: A Type 5 PDU was sent.
TARP P-A: Received TARP	TARP packet: A Type 3 PDU was received.
Lft	Lifetime of the PDU (in hops).
Seq	Sequence number of the PDU.
Prot type	Protocol type of the PDU.
URC	The update remote cache bit.

**Table 168      Debug TARP Field Descriptions—TARP Query Command (Continued)**

<b>Field</b>	<b>Descriptions</b>
Ttid len	Destination TID length.
Stid len	Source TID length.
Prot addr len	Protocol address length (bytes).
Packet sent/propagated	NSAP address of the router that sent or propagated the PDU.
Originator's NSAP	NSAP address that the PDU was sent from.
Originator's TID	TID that the PDU was sent from.
TARP-PA: Created new DYNAMIC cache entry	TARP packet: A dynamic entry was made to the local TID cache.

Related Command

**debug tarp events**

## debug tdm

Use the **debug tdm** EXEC command to display time division multiplexer (TDM) bus connection information each time a connection is made on the Cisco AS5200 access server. The **no** form of this command disables debugging output.

**[no] debug tdm**

### Usage Guidelines

Use the **debug tdm** command if you are losing channel data between the dual T1 Primary Rate Interfaces (PRI) and any termination points, such as an Ethernet or modem point.

This command displays the TDM bus connection information for each TDM device installed in the access server. One TDM device exists on the PRI card, on the motherboard, and on each modem card. Expect up to 256 TDM connections to be displayed on your terminal when this command is enabled.

### Sample Display

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

```
AS5200# debug tdm

dialtone connection requested.
TDM(reg: 0x2138100): Close connection to ST07, channel 1
TDM(reg: 0x2138100): Connect STi3, channel 1 to ST07, channel 1
```

## debug tftp

Use the **debug tftp** EXEC command to display Trivial File Transfer Protocol (TFTP) debugging information when encountering problems netbooting or using the **copy tftp system:running-config** or **copy system:running-config tftp** commands. The **no** form of this command disables debugging output.

**[no] debug tftp**

### Sample Display

The following is sample output from the **debug tftp** command from the EXEC command **copy system:running-config tftp**.

```
Router# debug tftp

TFTP: msclock 0x292B4; Sending write request (retry 0), socket_id 0x301DA8
TFTP: msclock 0x2A63C; Sending write request (retry 1), socket_id 0x301DA8
TFTP: msclock 0x2A6DC; Received ACK for block 0, socket_id 0x301DA8
TFTP: msclock 0x2A6DC; Received ACK for block 0, socket_id 0x301DA8
TFTP: msclock 0x2A6DC; Sending block 1 (retry 0), socket_id 0x301DA8
TFTP: msclock 0x2A6E4; Received ACK for block 1, socket_id 0x301DA8
```

Table 169 describes the significant fields in the first line of output.

**Table 169** Debug TFTP Field Descriptions

Message	Description
TFTP:	This entry describes a TFTP packet.
msclock 0x292B4;	Internal timekeeping clock (in milliseconds).
Sending write request (retry 0)	The TFTP operation.
socket_id 0x301DA8	Unique memory address for the socket for the TFTP connection.

## debug token ring

Use the **debug token ring EXEC** command to display messages about Token Ring interface activity. The **no** form of this command disables debugging output.

**[no] debug token ring**

### Usage Guidelines

This command reports several lines of information for each packet sent or received and is intended for low traffic, detailed debugging.

The Token Ring interface records provide information regarding the current state of the ring. These messages are only displayed when the **debug token events** command is enabled.

The **debug token ring** command invokes verbose Token Ring hardware debugging. This includes detailed displays as traffic arrives and departs the unit.

---

**Note** It is best to use this command only on router/bridges with light loads.

---

### Sample Display

The following is sample output from the **debug token ring** command:

```
Router# debug token ring

TR0: Interface is alive, phys. addr 5000.1234.5678
TR0: in: MAC: acfc: 0x1105 Dst: c000.ffff.ffff Src: 5000.1234.5678 bf: 0x45
TR0: in:   riflcn 0, rd_offset 0, llc_offset 40
TR0: out: MAC: acfc: 0x0040 Dst: 5000.1234.5678 Src: 5000.1234.5678 bf: 0x00
TR0: out: LLC: AAAA0300 00009000 00000100 AAC00000 00000802 50001234 ln: 28
TR0: in: MAC: acfc: 0x1140 Dst: 5000.1234.5678 Src: 5000.1234.5678 bf: 0x09
TR0: in: LLC: AAAA0300 00009000 00000100 AAC0B24A 4B4A6768 74732072 ln: 28
TR0: in:   riflcn 0, rd_offset 0, llc_offset 14
TR0: out: MAC: acfc: 0x0040 Dst: 5000.1234.5678 Src: 5000.1234.5678 bf: 0x00
TR0: out: LLC: AAAA0300 00009000 00000100 D1D00000 FE11E636 96884006 ln: 28
TR0: in: MAC: acfc: 0x1140 Dst: 5000.1234.5678 Src: 5000.1234.5678 bf: 0x09
TR0: in: LLC: AAAA0300 00009000 00000100 D1D0774C 4DC2078B 3D000160 ln: 28
TR0: in:   riflcn 0, rd_offset 0, llc_offset 14
TR0: out: MAC: acfc: 0x0040 Dst: 5000.1234.5678 Src: 5000.1234.5678 bf: 0x00
TR0: out: LLC: AAAA0300 00009000 00000100 F8E00000 FE11E636 96884006 ln: 28
```

Table 170 describes the significant fields in the second line of output.

**Table 170** Debug Token Ring Field Descriptions—Part 1

Message	Description
TR0:	Name of the interface associated with the Token Ring event.
in:	Indication of whether the packet was input to the interface (in) or output from the interface (out).
MAC:	The type of packet, as follows: <ul style="list-style-type: none"> <li>• MAC—Media Access Control</li> <li>• LLC—Link Level Control</li> </ul>
acfc: 0x1105	Access Control, Frame Control bytes, as defined by the IEEE 802.5 standard.
Dst: c000.ffff.ffff	Destination address of the frame.

**Table 170      Debug Token Ring Field Descriptions—Part 1 (Continued)**

<b>Message</b>	<b>Description</b>
Src: 5000.1234.5678	Source address of the frame.
bf: 0x45	Bridge flags for internal use by technical support staff.

Table 171 describes the significant fields shown in the third line of output.

**Table 171      Debug Token Ring Field Descriptions—Part 2**

<b>Message</b>	<b>Description</b>
TR0:	Name of the interface associated with the Token Ring event.
in:	Indication of whether the packet was input to the interface (in) or output from the interface (out).
riflen 0	Length of the RIF field (in bytes).
rd_offset 0	Offset (in bytes) of the frame pointing to the start of the RIF field.
llc_offset 40	Offset in the frame pointing to the start of the LLC field.

Table 172 describes the significant fields shown in the fifth line of output.

**Table 172      Debug Token Ring Field Descriptions—Part 3**

<b>Message</b>	<b>Description</b>
TR0:	Name of the interface associated with the Token Ring event.
out:	Indication of whether the packet was input to the interface (in) or output from the interface (out).
LLC:	The type of frame, as follows: <ul style="list-style-type: none"> <li>• MAC—Media Access Control</li> <li>• LLC—Link Level Control</li> </ul>
AAAA0300	This and the octets that follow it indicate the contents (hex) of the frame.
ln: 28	The length of the information field (in bytes).

## debug v120 event

Use the **debug v120 event** EXEC command to display information on V.120 activity. The **no** form of this command disables debugging output.

**[no] debug v120 event**

### Usage Guidelines

V.120 is an ITU specification that allows for reliable transport of synchronous, asynchronous, or bit transparent data over ISDN bearer channels.

For complete information on the V.120 process, use the **debug v120 packet** command along with the **debug v120 event** command. V.120 events are activity events rather than error conditions.

### Sample Display

The following is sample output from the **debug v120 event** command of V.120 starting up and stopping. Also included is the interface that V.120 is running on (BR 0) and where the V.120 configuration parameters are obtained from (default).

```
Router# debug v120 event

0:01:47: BR0:1-v120 started - Setting default V.120 parameters
0:02:00: BR0:1:removing v120
```

### Related Command

**debug v120 packet**

## debug v120 packet

Use the **debug v120 packet** EXEC command to display general information on all incoming and outgoing V.120 packets. The **no** form of this command disables debugging output.

**[no] debug v120 packet**

### Usage Guidelines

The **debug v120 packet** command shows every packet on the V.120 session. You can use this information to determine whether incompatibilities exist between Cisco’s V.120 implementation and other vendors’ V.120 implementations.

V.120 is an ITU specification that allows for reliable transport of synchronous, asynchronous, or bit transparent data over ISDN bearer channels.

For complete information on the V.120 process, use the **debug v120 events** command along with the **debug v120 packet** command.

### Sample Display

The following is sample output from the **debug v120 packet** command for a typical session startup.

```
Router# debug v120 packet

0:03:27: BR0:1: I SABME:11i 256 C/R 0 P/F=1
0:03:27: BR0:1: O UA:11i 256 C/R 1 P/F=1
0:03:27: BR0:1: O IFRAME:11i 256 C/R 0 N(R)=0 N(S)=0 P/F=0 len 43
0x83 0xD 0xA 0xD 0xA 0x55 0x73 0x65
0x72 0x20 0x41 0x63 0x63 0x65 0x73 0x73
0:03:27: BR0:1: I RR:11i 256 C/R 1 N(R)=1 P/F=0
0:03:28: BR0:1: I IFRAME:11i 256 C/R 0 N(R)=1 N(S)=0 P/F=0 len 2
0x83 0x63
0:03:28: BR0:1: O RR:11i 256 C/R 1 N(R)=1 P/F=0
0:03:29: BR0:1: I IFRAME:11i 256 C/R 0 N(R)=1 N(S)=1 P/F=0 len 2
0x83 0x31
0:03:29: BR0:1: O RR:11i 256 C/R 1 N(R)=2 P/F=0
%LINEPROTO-5-UPDOWN: Line protocol on Interface BRI0: B-Channel 1, changed state to up
0:03:31: BR0:1: I IFRAME:11i 256 C/R 0 N(R)=1 N(S)=2 P/F=0 len 2
0x83 0x55
0:03:32: BR0:1: I IFRAME:11i 256 C/R 0 N(R)=1 N(S)=3 P/F=0 len 3
0x83 0x31 0x6F
0:03:32: BR0:1: O RR:11i 256 C/R 1 N(R)=3 P/F=0
0:03:32: BR0:1: I IFRAME:11i 256 C/R 0 N(R)=1 N(S)=4 P/F=0 len 2
0x83 0x73
0:03:32: BR0:1: O RR:11i 256 C/R 1 N(R)=5 P/F=0
0:03:32: BR0:1: I IFRAME:11i 256 C/R 0 N(R)=1 N(S)=5 P/F=0 len 2
0x83 0xA
0:03:32: BR0:1: O IFRAME:11i 256 C/R 0 N(R)=6 N(S)=1 P/F=0 len 9
0x83 0xD 0xA 0x68 0x65 0x66 0x65 0x72 0x3E
```

Table 173 describes the significant fields in the display.

**Table 173** Debug V.120 Packet Field Descriptions

Field	Descriptions
BR0:1	Interface number associated with this debugging information.
I/O	Packet going into or out of the interface.

**Table 173**      **Debug V.120 Packet Field Descriptions (Continued)**

<b>Field</b>	<b>Descriptions</b>
SABME, UA, IFRAME, RR	V.120 packet type. In this case: <ul style="list-style-type: none"><li>• SABME—set asynchronous balanced mode, extended</li><li>• US—unnumbered acknowledgment</li><li>• IFRAME—information frame</li><li>• RR—receive ready</li></ul>
lli 256	Logical link identifier number.
C/R 0	Command or response.
P/F=1	Poll final.
N(R)=0	Number received.
N(S)=0	Number sent.
len 43	Number of data bytes in the packet.
0x83	Up to 16 bytes of data.

**Related Command****debug tarp events**

## debug vg-anylan

To monitor error information and 100VG connection activity, use the **debug vg-anylan EXEC** command. The **no** form of this command disables debugging output.

**[no] debug vg-anylan**

### Usage Guidelines

This command could create large amounts of command output.

### Sample Display

The following is sample output from the **debug vg-anylan** command:

```
Router# debug vg-anylan

%HP100VG-5-LOSTCARR: HP100VG(2/0), lost carrier
```

Table 174 lists the possible messages that could be generated by this command.

**Table 174 Debug VG-AnyLAN Message Descriptions**

Message	Description	Action
%HP100VG-5-LOSTCARR: HP100VG(2/0), lost carrier	Lost carrier debug message. The VG controller detects that the link to the hub is down due to cable, hub, or VG controller problem.	Check, repair, or replace the cable or hub. If you determine that the cable and hub are functioning normally, repair or replace the 100VG-AnyLAN port adapter.
%HP100VG-5-CABLEERR: HP100VG(2/0), cable error, training failed	Bad cable error messages. Cable did not pass training. <sup>1</sup>	Check, repair, or replace the cable or hub. If you determine that the cable and hub are functioning normally, repair or replace the 100VG-AnyLAN port adapter.
%HP100VG-5-NOCABLE: HP100VG(2/0), no tone detected, check cable, hub	No cable attached error message. The VG MAC cannot hear tones from the hub. <sup>1</sup>	Check, repair, or replace the cable or hub. If you determine that the cable and hub are functioning normally, repair or replace the 100VG-AnyLAN port adapter.
HP100VG-1-FAIL: HP100VG(2/0), Training Fail - unable to login to the hub	Training to the VG network failed. Login to the hub rejected by the hub. <sup>1</sup>	Take action based on the following error messages: <ul style="list-style-type: none"> <li>• %HP100VG-1-DUPMAC: HP100VG(2/0), A duplicate MAC address has been detected</li> <li>• HP100VG-1-LANCNF: HP100VG(2/0), Configuration is not compatible with the network</li> <li>• %HP100VG-1-ACCESS: HP100VG(2/0), Access to network is not allowed</li> </ul>
%HP100VG-1-DUPMAC: HP100VG(2/0), A duplicate MAC address has been detected	Duplicate MAC address on the same VG network. Two VG devices on the same LAN segment have the same MAC address.	Check the router configuration to make sure that no duplicate MAC address is configured.

**Table 174 Debug VG-AnyLAN Message Descriptions (Continued)**

<b>Message</b>	<b>Description</b>	<b>Action</b>
%HP100VG-1-LANCF: HP100VG(2/0), Configuration is not compatible with the network	Configuration of the router is not compatible to the network.	Check that the configuration of the hub for Frame Format, Promiscuous, and Repeater bit indicates the proper configuration.
%HP100VG-1-ACCESS: HP100VG(2/0), Access to network is not allowed	Access to the VG network is denied by the hub.	Check the configuration of the hub.
%HP100VG-3-NOTHP100V G: Device reported 0x5101A	Could not find the 100VG PCI device on a 100VG-AnyLAN port adapter.	Make sure the 100VG-AnyLAN port adapter is properly seated in the slot. Otherwise repair or replace the 100VG-AnyLAN port adapter.
%HP100VG-1-DISCOVER: Only found 0 interfaces on bay 2, shutting down bay	No 100VG interface detected on a 100VG-AnyLAN port adapter in a slot.	Make sure the 100VG-AnyLAN port adapter is properly seated in the slot. Otherwise repair or replace the 100VG-AnyLAN port adapter.

- 1 This message might display when the total load on the cascaded hub is high. Wait at least 20 seconds before checking to see if the training really failed. Check if the protocol is up after 20 seconds before starting troubleshooting.

## debug vines arp

Use the **debug vines arp** EXEC command to display debugging information on all Virtual Integrated Network Service (VINES) Address Resolution Protocol (ARP) packets that the router sends or receives. The **no** form of this command disables debugging output.

**[no] debug vines arp**

### Sample Display

The following is sample output from the **debug vines arp** command:

```
Router# debug vines arp

VNSARP: received ARP type 0 from 0260.8c43.a7e4
VNSARP: sending ARP type 1 to 0260.8c43.a7e4
VNSARP: received ARP type 2 from 0260.8c43.a7e4
VNSARP: sending ARP type 3 to 0260.8c43.a7e4 assigning address 3001153C:8004
VSARP: received ARP type 0 from 0260.8342.1501
VSARP: sending ARP type 1 to 0260.8342.1501
VSARP: received ARP type 2 from 0260.8342.1501
VSARP: sending ARP type 3 to 0260.8342.1501 assigning address 3001153C:8005,
sequence 143C, metric 2
```

In the sample output, the first four lines show a non-sequenced ARP transaction and the second four lines show a sequenced ARP transaction. Within the first group of four lines, the first line shows that the router received an ARP request (type 0) from indicated station address 0260.8c43.a7e4. The second line shows that the router is sending back the ARP service response (type 1), indicating that it is willing to assign VINES Internet addresses. The third line shows that the router received a VINES Internet address assignment request (type 2) from address 0260.8c43.a7e4. The fourth line shows that the router is responding (type 3) to the address assignment request from the client and assigning it the address 3001153C:8004.

Within the second group of four lines, the sequenced ARP packet also includes the router' current sequence number and the metric value between the router and the client.

Table 175 describes significant fields in the display.

**Table 175 Debug VINES ARP Field Descriptions**

Field	Description
VNSARP:	Banyan VINES nonsequenced ARP message.
VSARP:	Banyan VINES sequenced ARP message.
received ARP type 0	ARP request of type 0 was received. Possible type values follow: <ul style="list-style-type: none"> <li>• 0—Query request. The ARP client broadcasts a type 0 message to request an ARP service to respond.</li> <li>• 1—Service response. The ARP service responds with a type 1 message to an ARP client's query request.</li> <li>• 2—Assignment request. The ARP client responds to a service response with a type 2 message to request a VINES Internet address.</li> <li>• 3—Assignment response. The ARP service responds to an assignment request with a type 3 message that includes the assigned VINES Internet address.</li> </ul>
from 0260.8c43.a7e4	Indicates the source address of the packet.

## debug vines echo

Use the **debug vines echo** EXEC command to display information on all MAC-level echo packets that the router sends or receives. Banyan VINES interface testing programs make use of these echo packets. The **no** form of this command disables debugging output.

**[no] debug vines echo**

---

**Note** These echo packets do not include network layer addresses.

---

### Sample Display

The following is sample output from the **debug vines echo** command:

```
Router# debug vines echo
VINESECHO: 100 byte packet from 0260.8c43.a7e4
```

Table 176 describes the significant fields in the display.

**Table 176**      **Debug VINES Echo Field Descriptions**

<b>Field</b>	<b>Description</b>
VINESECHO	Indication that this is a <b>debug vines echo</b> message.
100 byte packet	Packet size in bytes.
from 0260.8c43.a7e4	Source address of the echo packet.

## debug vines ipc

Use the **debug vines ipc** EXEC command to display information on all transactions that occur at the VINES IPC layer, which is one of the two VINES transport layers. The **no** form of this command disables debugging output.

**[no] debug vines ipc**

### Usage Guidelines

You can use the **debug vines ipc** command to discover why an IPC layer process on the router is not communicating with another IPC layer process on another router or Banyan VINES server.

### Sample Display

The following is sample output from the **debug vines ipc** command for three pairs of transactions. For more information about these fields or their values, refer to Banyan VINES documentation.

```
Router# debug vines ipc

VIPIC: sending IPC Data to Townsaver port 7 from port 7
       r_cid 0, l_cid 1, seq 1, ack 0, length 12
VIPIC: received IPC Data from Townsaver port 7 to port 7
       r_cid 51, l_cid 1, seq 1, ack 1, length 32
VIPIC: sending IPC Ack to Townsaver port 0 from port 0
       r_cid 51, l_cid 1, seq 1, ack 1, length 0
```

Table 177 describes the significant fields in the display.

**Table 177**      **Debug VINES IPC Field Descriptions**

Field	Description
VIPIC:	Indicates that this is output from the <b>debug vines ipc</b> command.
sending	Indicates that the router is either sending an IPC packet to another router or has received an IPC packet from another router.
IPC Data to	Indicates the type of IPC frame: <ul style="list-style-type: none"> <li>• Acknowledgment</li> <li>• Data</li> <li>• Datagram</li> <li>• Disconnect</li> <li>• Error</li> <li>• Probe</li> </ul>
Townsaver port 7	Indicates the machine name as assigned using the VINES <b>host</b> command, or IP address of the other router. Also indicates the port on that machine through which the packet has been transmitted.
from port 7	Indicates the port on the router through which the packet has been transmitted.
r_cid 0, l_cid 1, seq 1, ack 0, length 12	Indicates the values for various fields in the IPC layer header of this packet. Refer to Banyan VINES documentation for more information.

## debug vines netrpc

Use the **debug vines netrpc** EXEC command to display information on all transactions that occur at the VINES NetRPC layer, which is the VINES Session/Presentation layer. The **no** form of this command disables debugging output.

**[no] debug vines netrpc**

### Usage Guidelines

You can use the **debug vines netrpc** command to discover why a NetRPC layer process on the router is not communicating with another NetRPC layer process on another router or Banyan server.

### Sample Display

The following is sample output from the **debug vines netrpc** command. For more information about these fields or their values, refer to Banyan VINES documentation.

```
Router# debug vines netrpc

VRPC: sending RPC call to Townsaver
VRPC: received RPC return from Townsaver
```

Table 178 describes the fields shown in the first line of output.

**Table 178**      **Debug VINES NetRPC Field Descriptions**

Field	Description
VRPC:	Indicates that this is output from the <b>debug vines netrpc</b> command.
sending RPC	Indicates that the router is either sending a NetRPC packet to another router or has received a NetRPC packet from another router.
call	Indicates the transaction type: <ul style="list-style-type: none"> <li>• abort</li> <li>• call</li> <li>• reject</li> <li>• return</li> <li>• return address</li> <li>• search</li> <li>• search all</li> </ul>
Townsaver	Indicates the machine name as assigned using the VINES <b>host</b> command or IP address of the other router.

## debug vines packet

Use the **debug vines packet** EXEC command to display general VINES debugging information. This information includes packets received, generated, and forwarded, as well as failed access checks and other operations. The **no** form of this command disables debugging output.

**[no] debug vines packet**

### Sample Display

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

```
Router# debug vines packet

VINES: s=30028CF9:1 (Ether2), d=FFFFFFFF:FFFF, rcvd w/ hops 0
VINES: s=3000CBD4:1 (Ether1), d=3002ABEA:1 (Ether2), g=3002ABEA:1, sent
VINES: s=3000CBD4:1 (Ether1), d=3000B959:1, rcvd by gw
VINES: s=3000B959:1 (local), d=3000CBD4:1 (Ether1), g=3000CBD4:1, sent
```

Table 179 describes the fields shown in the first line of output.

**Table 179**      **Debug VINES Packet Field Descriptions**

Field	Description
VINES:	Indicates that this is a Banyan VINES packet.
s = 30028CF9:1 (Ether2)	Indicates source address of the packet. Indicates the interface through which the packet was received.
d = FFFFFFFF:FFFF	Indicates that the destination is a broadcast address.
rcvd w/ hops 0	Indicates that the packet was received because it was a local broadcast packet. The remaining hop count in the packet was zero (0).

In the following line, the destination is the address 3002ABEA:1 associated with interface Ether2. Source address 3000CBD4:1 sent a packet to this destination through the gateway at address 3000ABEA:1.

```
VINES: s=3000CBD4:1 (Ether1), d=3002ABEA:1 (Ethernet2), g=3002ABEA:1, sent
```

In the following line, the router being debugged is the destination address (3000B959:1):

```
VINES: s=3000CBD4:1 (Ether1), d=3000B959:1, rcvd by gw
```

In the following line, (local) indicates that the router being debugged generated the packet:

```
VINES: s=3000B959:1 (local), d=3000CBD4:1 (Ether1), g=3000CBD4:1, sent
```

## debug vines routing

Use the **debug vines routing** EXEC command to display information on all VINES RTP update messages sent or received and all routing table activities that occur in the router. The **no** form of this command disables debugging output.

**[no] debug vines routing [verbose]**

### Syntax Description

**verbose** (Optional) Provides detailed information about the contents of each update.

### Sample Displays

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

```
router# debug vines routing
VSRTTP: generating change update, sequence number 0002C791
Update sent - VSRTTP: sent update to Broadcast on Hssi0
Update received - VSRTTP: received update from LabRouter on Hssi0
VSRTTP: LabRouter-Hs0-HDLC up -> up, change update, onemore
VRTTP: sending update to Broadcast on Ethernet0
VSRTTP: generating null update
VSRTTP: Sending update to Aloe on Hssi0
```

S2854

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

```
Router# debug vines routing verbose
VRTTP: sending update to Broadcast on Ethernet0
network 30011E7E, metric 0020 (0.4000 seconds)
network 30015800, metric 0010 (0.2000 seconds)
network 3003148A, metric 0020 (0.4000 seconds)
VSRTTP: generating change update, sequence number 0002C795
network Router9 metric 0010, seq 00000000, flags 09
network RouterZZ metric 0230, seq 00052194, flags 02
VSRTTP: sent update to Broadcast on Hssi0
VSRTTP: received update from LabRouter on Hssi0
update: type 00, flags 07, id 000E, ofst 0000, seq 15DFC, met 0010
network LabRouter from the server
network Router9 metric 0020, seq 00000000, flags 09
VSRTTP: LabRouter-Hs0-HDLC up -> up, change update, onemore
```

The output describes two VINES routing updates; the first includes two entries and the second includes three entries. Explanations for selected lines follow.

The following line shows that the router sent a periodic routing update to the broadcast address FFFFFFFF:FFFF through the Ethernet0 interface:

```
VRTTP: sending update to Broadcast on Ethernet0
```

The following line indicates that the router knows how to reach network 30011E7E, which is a metric of 0020 away from the router. The value that follows the metric (0.4000 seconds) interprets the metric in seconds.

```
network 30011E7E, metric 0020 (0.4000 seconds)
```

The following lines show that the router sent a change routing update to the Broadcast addresses on the Hssi0 interface using the Sequenced Routing Update Protocol (SRTP) routing protocol:

```
VS RTP: generating change update, sequence number 0002C795
VS RTP: Sending update to Broadcast on Hssi0
```

The lines in between the previous two indicate that the router knows how to reach network Router9, which is a metric of 0010 (0.2000 seconds) away from the router. The sequence number for Router9 is zero, and according to the 0x08 bit in the flags field, is invalid. The 0x01 bit of the flags field indicates that Router9 is attached via a LAN interface.

```
network Router9          metric 0010, seq 00000000, flags 09
```

The next lines indicate that the router can reach network RouterZZ, which is a metric of 0230 (7.0000 seconds) away from the router. The sequence number for RouterZZ is 0052194. The 0x02 bit of the flags field indicates that RouterZZ is attached via a WAN interface.

```
network RouterZZ        metric 0230, seq 00052194, flags 02
```

The following line indicates that the router received a routing update from the router LabRouter through the Hssi0 interface:

```
VS RTP: received update from LabRouter on Hssi0
```

The following line displays all SRTP values contained in the header of the SRTP packet. This is a type 00 packet, which is a routing update, and the flags field is set to 07, indicating that this is a change update (0x04) and contains both the beginning (0x01) and end (0x02) of the update. This overall update is update number 000E from the router, and this fragment of the update contains the routes beginning at offset 0000 of the update. The sending router's sequence number is currently 00015DFC, and its configured metric for this interface is 0010.

```
update: type 00, flags 07, id 000E, ofst 0000, seq 00015DFC, met 0010
```

The following line implies that the server sending this update is directly accessible to the router (even though VINES servers do not explicitly list themselves in routing updates). Because this is an implicit entry in the table, the other information for this entry is taken from the previous line.

```
network LabRouter from the server
```

As the first actual entry in the routing update from LabRouter, the following line indicates that Router9 can be reached by sending to this server. This network is a metric of 0020 away from the sending server.

```
network Router9          metric 0020, seq 00000000, flags 09
```

## debug vines service

Use the **debug vines service** EXEC command to display information on all transactions that occur at the VINES Service (or applications) layer. The **no** form of this command disables debugging output.

**[no] debug vines service**

### Usage Guidelines

You can use the **debug vines service** command to discover why a VINES Service layer process on the router is not communicating with another Service layer process on another router or Banyan server.

---

**Note** Because the **debug vines service** command provides the highest level overview of VINES traffic through the router, it is best to begin debugging using this command, and then proceed to use lower-level VINES **debug** commands as necessary.

---

### Sample Display

The following is sample output from the **debug vines service** command:

```
router# debug vines service
```

Sent/ Response pair	<pre>VSRV: Get Time Info sent to Townsaver VSRV: Get Time Info response from Townsaver, time: 01:47:54 PDT Apr 29 1993 VSRV:      epoch SS@Aloe@Servers-10, age: 0:15:15</pre>	S2565
---------------------------	--	-------

As the sample suggests, **debug vines service** lines of output appear as activity pairs—either a sent/response pair as shown, or as a received/sent pair.

Table 180 describes the fields shown in the second line of output. For more information about these fields or their values, refer to Banyan VINES documentation.

**Table 180**      **Debug VINES Service Field Descriptions—Part 1**

Field	Description
VSRV:	Indicates that this is output from the <b>debug vines service</b> command.
Get Time Info	Indicates one of three packet types: <ul style="list-style-type: none"> <li>• Get Time Info</li> <li>• Time Set</li> <li>• Time Sync</li> </ul>
response from	Indicates whether the packet was sent to another router, a response from another router, or received from another router.
Townsaver	Indicates the machine name as assigned using the VINES <b>host</b> command, or IP address of the other router.
time: 01:47:54 PDT Apr 29 1993	Indicates the current time in hours:minutes:seconds and current date.

Table 181 describes the fields shown in the third line of output. This line is an extension of the first two lines of output. For more information about these fields or their values, refer to Banyan VINES documentation.

**Table 181      Debug VINES Service Field Descriptions—Part 2**

<b>Field</b>	<b>Description</b>
VSRV:	Output from the <b>debug vines service</b> command.
epoch	Line of output that describes a VINES epoch.
SS@Aloe@Servers-10	Epoch name.
age: 0:15:15	Epoch—elapsed time since the time was last set in the network.

## debug vines state

Use the **debug vines state** EXEC command to display information on the VINES SRTP state machine transactions. The **no** form of this command disables debugging output.

**[no] debug vines state**

### Usage Guidelines

This command provides a subset of the information provided by the **debug vines routing** command, showing only the transactions made by the SRTP state machine. Refer to the **debug vines routing** command for descriptions of output from the **debug vines state** command.

## debug vines table

Use the **debug vines table** EXEC command to display information on all modifications to the VINES routing table. The **no** form of this command disables debugging output.

**[no] debug vines table**

### Usage Guidelines

This command provides a subset of the information produced by the **debug vines routing** command, as well as some more detailed information on table additions and deletions.

### Sample Display

The following is sample output from the **debug vines table** command:

```
Router# debug vines table  
  
VINESRTP: create neighbor 3001153C:8004, interface Ethernet0
```

Table 182 describes the significant fields in the display.

**Table 182**      **Debug VINES Table Field Descriptions**

Field	Description
VINESRTP:	Indicates that this is a <b>debug vines routing</b> or <b>debug vines table</b> message.
create neighbor 3001153C:8004	Indicates that the client at address 3001153C:8004 has been added to the Banyan VINES neighbor table.
interface Ethernet 0	Indicates that this neighbor can be reached through the router interface named Ethernet0.

## debug vlan packet

Use the **debug vlan packet** EXEC command to display general information on virtual LAN (VLAN) packets that the router received but is not configured to support. The **no** form of this command disables debugging output.

**[no] debug vlan packet**

### Usage Guidelines

The **debug vlan packet** command displays only packets with a VLAN identifier that the router is not configured to support. This command allows you to identify other VLAN traffic on the network. Virtual LAN packets that the router is configured to route or switch are counted and indicated when you use the **show vlans** command.

### Sample Display

The following is sample output from the **debug vlan packet** output. In this example, a VLAN packet with a VLAN ID of 1000 was received on FDDI 0 interface and this interface was not configured to route or switch this VLAN packet.

```
Router# debug vlan packet

vLAN: IEEE 802.10 packet bearing vLAN ID 1000 received on interface
      Fddi0 which is not configured to route/switch ID 1000.
```

## debug voice all

Use the **debug voice all** EXEC command to display debugging information for all components of the Voice Call Manager. The **no** form of this command disables debugging output.

[no] **debug voice all** [*slot/port*]

### Syntax Description

<i>slot/port</i>	(Optional) The <i>slot/port</i> number of the voice port. If the <i>slot/port</i> is entered, then only debugging information for that voice port is displayed. If the <i>slot/port</i> is not entered, debugging information for all voice ports is displayed.
------------------	---

### Usage Guidelines

This command is valid on the Cisco MC3810 only.

### Sample Display

The **debug voice all** output provides debug output for all the debug commands for the Voice Call Manager compiled into one display. For sample output of the individual commands, see the sample displays for the **debug voice cp**, **debug voice eecm**, **debug voice protocol**, **debug voice signaling**, and **debug voice tds** commands.

### Related Commands

- debug voice cp**
- debug voice eecm**
- debug voice protocol**
- debug voice signaling**
- debug voice tds**

## debug voice cp

Use the **debug voice cp** EXEC command to display debugging information for the Voice Call Processing State Machine. The **no** form of this command disables debugging output.

**[no] debug voice cp** [*slot/port*]

### Syntax Description

*slot/port* (Optional) The slot/port number of the voice port. If the *slot/port* is entered, then only debugging information for that voice port is displayed.

### Usage Guidelines

This command is valid on the Cisco MC3810 only.

### Sample Display

The following is sample output from the **debug voice cp** command:

```
router# debug voice cp 1/1

Voice Call Processing State Machine debugging is on

1/1: CPD( ), idle gets event seize_ind
1/1: CPD( ), idle gets event dsp_ready
1/1: CPD( ), idle ==> collect
1/1: CPD(in), collect gets event digit
1/1: CPD(in), collect gets event digit
1/1: CPD(in), collect gets event digit
1/1: CPD(in), collect gets event digit
1/1: CPD(in), collect gets event addr_done
1/1: CPD(in), collect ==> request
1/1: CPD(in), request gets event call_proceeding
1/1: CPD(in), request ==> in_wait_answer
1/1: CPD(in), in_wait_answer gets event call_accept
1/1: CPD(in), in_wait_answer gets event call_answered
1/1: CPD(in), in_wait_answer ==> connected
1/1: CPD(in), connected gets event peer_onhook
1/1: CPD(in), connected ==> disconnect_wait
1/1: CPD(in), disconnect_wait gets event idle_ind
1/1: CPD(in), disconnect_wait ==> idle
```

### Related Commands

**debug voice all**  
**debug voice eecm**  
**debug voice protocol**  
**debug voice signaling**  
**debug voice tds**

## debug voice eecm

Use the **debug voice eecm** EXEC command to display debugging information for the Voice End-to-End Call Manager. The **no** form of this command disables debugging output.

**[no] debug voice eecm** [*slot/port*]

### Syntax Description

*slot/port* (Optional) The slot/port number of the voice port. If the *slot/port* is entered, then only debugging information for that voice port is displayed.

### Usage Guidelines

This command is valid on the Cisco MC3810 only.

### Sample Display

The following is sample output from the **debug voice eecm** command:

```
router# debug voice eecm

1/1: EECM(in), ST_NULL          EV_ALLOC_DSP
1/1: EECM(in), ST_DIGIT_COLLECT EV_PARSE_DIGIT 3
1/1: EECM(in), ST_DIGIT_COLLECT EV_PARSE_DIGIT 7
1/1: EECM(in), ST_DIGIT_COLLECT EV_PARSE_DIGIT 0
1/1: EECM(in), ST_DIGIT_COLLECT EV_PARSE_DIGIT 2
1/1: EECM(in), ST_ADDRESS_DONE  EV_OUT_SETUP
-1/-1: EECM(out), ST_NULL      EV_IN_SETUP
1/1: EECM(in), ST_OUT_REQUEST   EV_IN_PROCEED
1/2: EECM(out), ST_SEIZE        EV_ALLOC_DSP
1/2: EECM(out), ST_SEIZE        EV_OUT_ALERT
1/1: EECM(in), ST_OUT_REQUEST   EV_IN_ALERT
1/1: EECM(in), ST_OUT_REQUEST   EV_OUT_ALERT_ACK
1/2  EECM(out), ST_IN_PENDING   EV_OUT_CONNECT
1/1: EECM(in), ST_WAIT_FOR_ANSWER EV_IN_CONNECT
1/2: EECM(out), ST_ACTIVE       EV_OUT_REL
1/1: EECM(in), ST_ACTIVE        EV_IN_REL
1/1: EECM(in), ST_DISCONN_PENDING EV_OUT_REL_ACK
```

### Related Commands

**debug voice all**  
**debug voice cp**  
**debug voice protocol**  
**debug voice signaling**  
**debug voice tds**

## debug voice protocol

Use the **debug voice protocol** EXEC command to display debugging information for the Voice Line protocol State machine. The **no** form of this command disables debugging output.

**[no] debug voice protocol** [*slot/port*]

### Syntax Description

*slot/port* (Optional) The slot/port number of the voice port. If the *slot/port* is entered, then only debugging information for that voice port is displayed.

### Usage Guidelines

In the debugging display, the following abbreviations are used for the different signaling protocols:

LFXS	FXS trunk loop start protocol
LFXO	FXO trunk loop start protocol
GFXS	FXS trunk ground start protocol
GFXO	FXO trunk ground start protocol
E&M	E&M trunk protocol

This command is valid on the Cisco MC3810 only.

### Sample Display

The following is sample output from the **debug voice protocol** command:

```
router# debug voice protocol

Voice Line protocol State machine debugging is on

1/1: LFXS( ), idle gets event offhook
1/1: LFXS( ), idle ==> seize
1/1: LFXS(in), seize gets event ready
1/1: LFXS(in), seize ==> dial_tone
1/1: LFXS(in), dial_tone gets event digit
1/1: LFXS(in), dial_tone ==> collect
1/1: LFXS(in), collect gets event digit
1/1: LFXS(in), collect gets event digit
1/1: LFXS(in), collect gets event digit
1/1: LFXS(in), collect gets event addr_done
1/1: LFXS(in), collect ==> call_progress
1/2: LFXS( ), idle gets event seize
1/2: LFXS( ), idle ==> ringing
1/2: LFXS(out), ringing gets event dial_tone
1/2: LFXS(out), ringing gets event offhook
1/2: LFXS(out), ringing ==> connected
1/1: LFXS(in), call_progress gets event answer
1/1: LFXS(in), call_progress ==> connected
1/2: LFXS(out), connected gets event onhook
1/2: LFXS(out), connected ==> disconnect_wait
```

```
1/2: LFXS(out), disconnected_wait gets event disconnect
1/2: LFXS(out), disconnect_wait ==> cpc
1/1: LFXS(in), connected gets event disconnect
1/2: LFXS(out), connected ==> cpc
1/2: LFXS(out), cpc gets event offhook
1/2: LFXS(out), cpc gets event timer1
1/2: LFXS(out), cpc ==> cpc_recover
1/2: LFXS(out), cpc gets event timer1
1/2: LFXS(out), cpc_recover ==> offhook_wait
1/1: LFXS(in), offhook_wait gets event onhook
1/1: LFXS(in), offhook_wait ==> idle
1/2: LFXS(out), offhook_wait gets event onhook
1/2: LFXS(out), offhook_wait ==> idle
```

### Related Commands

- debug voice all**
- debug voice cp**
- debug voice eecm**
- debug voice signaling**
- debug voice tds**

## debug voice signaling

Use the **debug voice signaling** EXEC command to display debugging information for the voice port signaling. The **no** form of this command disables debugging output.

**[no] debug voice signaling** [*slot/port*]

### Syntax Description

*slot/port* (Optional) The slot/port number of the voice port. If the *slot/port* is entered, then only debugging information for that voice port is displayed.

### Usage Guidelines

This command is valid on the Cisco MC3810 only.

### Sample Display

The following is sample output from the **debug voice signaling** command:

```
router# debug voice signaling

1/1: TIU, report_local_hook=1
1/2: TIU, set ring cadence=1
1/2: TIU, ringer on
1/2: TIU, ringer off
1/2: TIU, ringer on
1/2: TIU, report_local_hook=1
1/2: TIU, turning off ringer due to SW ringtrip
1/2: TIU, ringer off
1/2: TIU, set ring cadence=0
1/2: TIU, ringer off
1/2: TIU, set reverse battery=1
1/2: TIU, set reverse battery=1
1/1: TIU, report_local_hook=0
1/2: TIU, set reverse battery=0
1/2: TIU, set loop disabled=1
1/1: TIU, set reverse battery=0
1/1: TIU, set loop disabled=1
1/2: TIU, report_local_hook=1
1/1: TIU, report_lead_gnd grounded=1
1/1: TIU, report_lead_gnd grounded=0
1/2: TIU, set loop disabled=0
1/1: TIU, set loop disabled=0
1/1: TIU, report_local_hook=0
1/2: TIU, report_local_hook=0
1/1: TIU, report_local_hook=1
1/2: TIU, report_local_hook=1
1/1: TIU, report_local_hook=0
1/2: TIU, report_local_hook=0
1/1: TIU, set reverse battery=0
1/2: TIU, set reverse battery=0
```

Related Commands

- debug voice all**
- debug voice cp**
- debug voice eecm**
- debug voice protocol**
- debug voice tds**

## debug voice tdsd

Use the **debug voice tdsd** EXEC command to display debugging information for the voice tandem switch. The **no** form of this command disables debugging output.

**[no] debug voice tdsd** [*slot/port*]

### Syntax Description

*slot/port* (Optional) The slot/port number of the voice port. If the *slot/port* is entered, then only debugging information for that voice port is displayed.

### Usage Guidelines

This command is valid on the Cisco MC3810 only.

### Sample Display

The following is sample output from the **debug voice tdsd** command:

```
router# debug voice tdsd

Voice tandem switch debugging is on

-1/-1: TDSM(out), ref= -1, state NULL gets event OUT_SETUP
1/1: TDSM(in), ref=6, state CALL_INITIATED gets event IN_CALLPROC
1/1: TDSM(in), ref=6, state OUTG_CALLPROC gets event IN_ALERTING
1/1: TDSM(in), ref=6, state CALL_DELIVERED gets event IN_CONNECT
1/1: TDSM(out),ref=6, state CALL_ACTIVE send out conn. ack
1/1: TDSM(out),ref=6, state CALL_ACTIVE send out release, cause LOCAL_ONHOOK
1/1: TDSM(in), ref=6, state RELEASE_REQ gets event IN_REL_COMP, cause REMOTE_ONHOOK
-1/-1: TDSM(in), ref=-1, state NULL gets event IN_SETUP
-1/-1: TDSM(out), ref=6, state INC_CALLPROC gets event OUT_ALERTING
1/1: TDSM(out),ref=6, state CALL_RECEIVED gets event OUT_CONNECT
1/1: TDSM(in), ref=6, state CONNECT_REQ gets event IN_CONN_ACK
1/1: TDSM(out),ref=6, state CALL_ACTIVE send out release, cause LOCAL_ONHOOK
1/1: TDSM(in), ref=6, state RELEASE_REQ gets event IN_REL_COMP, cause REMOTE_ONHOOK
-1/-1:TDSM(out), ref=-1, state NULL gets event OUT_SETUP
1/1: TDSM(in), ref=7, state CALL_INITIATED gets event IN_CALLPROC
1/1: TDSM(in), ref=7, state OUTG_CALLPROC gets event IN_ALERTING
1/1: TDSM(in), ref=7, state CALL_DELIVERED gets event IN_CONNECT
1/1: TDSM(out),ref=7, state CALL_ACTIVE send out conn.ack
1/1: TDSM(out),ref=7, state CALL_ACTIVE send out release, cause LOCAL_ONHOOK
-1/-1: TDSM(in), ref=-1, state NULL gets event IN_SETUP
-1/-1: TDSM(out), ref=7, state INC_CALLPROC gets event OUT_ALERTING
1/1: TDSM(out),ref=7. state CALL_RECEIVED gets event OUT_CONNECT
1/1: TDSM(in), ref=7, state CONNECT_REQ gets event IN_CONN_ACK
1/1: TDSM(in), ref=7, state CALL_ACTIVE send out release, cause LOCAL_ONHOOK
1/1: TDSM(in), ref=7, state RELEASE_REQ gets event IN_REL_COMP, cause REMOTE_ONHOOK
-1/-1: TDSM(out), ref=-1, state NULL gets event OUT_SETUP
1/1: TDSM(in), ref=8, state CALL_INITIATED gets event IN_CALLPROC
1/1: TDSM(in), ref=8, state OUTG_CALLPROC gets event IN_ALERTINGbug all
```

Related Commands

- debug voice all**
- debug voice cp**
- debug voice eecm**
- debug voice protocol**
- debug voice signaling**

## debug voip ccapi error

Use the **debug voip ccapi error** EXEC command to trace error logs in the call control API. The **no** form of this command disables debugging output.

**[no] debug voip ccapi error**

### Usage Guidelines

The **debug voip ccapi error** EXEC command traces the error logs in the call control API. Error logs are generated during normal call processing, when there are insufficient resources, or when there are problems in the underlying network-specific code, the higher call session application, or the call control API itself.

This debug command shows error events or unexpected behavior in system software. In most cases, no events will be generated.

## debug voip ccapi inout

Use the **debug voip ccapi inout** EXEC command to trace the execution path through the call control API. The **no** form of this command disables debugging output.

**[no] debug voip ccapi inout**

### Usage Guidelines

The **debug voip ccapi inout** EXEC command traces the execution path through the call control API, which serves as the interface between the call session application and the underlying network-specific software. You can use the output from this command to understand how calls are being handled by the router.

This command shows how a call flows through the system. Using this debug level, you can see the call setup and teardown operations performed on both the telephony and network call legs.

### Sample Display

The following output shows the call setup indicated and accepted by the router:

```
router# debug voip ccapi inout
cc_api_call_setup_ind (vdbPtr=0x60BFB530, callInfo={called=, calling=, fdest=0},
callID=0x60BFAEB8)
cc_process_call_setup_ind (event=0x60B68478)
sess_appl: ev(14), cid(1), disp(0)
ccCallSetContext (callID=0x1, context=0x60A7B094)
ccCallSetPeer (callID=0x1, peer=0x60C0A868, voice_peer_tag=2, encapType=1,
dest_pat=+14085231001, answer=)
ccCallSetupAck (callID=0x1)
```

The following output shows the caller entering DTMF digits until a dial-peer is matched:

```
cc_api_call_digit (vdbPtr=0x60BFB530, callID=0x1, digit=4, mode=0)
sess_appl: ev(8), cid(1), disp(0)
ssa: cid(1)st(0)oldst(0)cfid(-1)csize(0)in(1)fDest(0)
cc_api_call_digit (vdbPtr=0x60BFB530, callID=0x1, digit=1, mode=0)
sess_appl: ev(8), cid(1), disp(0)
ssa: cid(1)st(0)oldst(0)cfid(-1)csize(0)in(1)fDest(0)
cc_api_call_digit (vdbPtr=0x60BFB530, callID=0x1, digit=0, mode=0)
sess_appl: ev(8), cid(1), disp(0)
ssa: cid(1)st(0)oldst(0)cfid(-1)csize(0)in(1)fDest(0)
cc_api_call_digit (vdbPtr=0x60BFB530, callID=0x1, digit=0, mode=0)
sess_appl: ev(8), cid(1), disp(0)
ssa: cid(1)st(0)oldst(0)cfid(-1)csize(0)in(1)fDest(0)
cc_api_call_digit (vdbPtr=0x60BFB530, callID=0x1, digit=1, mode=0)
sess_appl: ev(8), cid(1), disp(0)
ssa: cid(1)st(0)oldst(0)cfid(-1)csize(0)in(1)fDest(0)
ccCallProceeding (callID=0x1, prog_ind=0x0)
ssaSetupPeer cid(1), destPat(+14085241001), matched(8), prefix(), peer(60C0E710)
```

The following output shows the call setup over the IP network to the remote router:

```
ccCallSetupRequest (peer=0x60C0E710, dest=, params=0x60A7B0A8 mode=0,
*callID=0x60B6C110)
ccIFCallSetupRequest: (vdbPtr=0x60B6C5D4, dest=, callParams={called=+14085241001,
calling=+14085231001, fdest=0, voice_peer_tag=104}, mode=0x0)
ccCallSetContext (callID=0x2, context=0x60A7B2A8)
```

The following output shows the called party is alerted, a codec is negotiated, and voice path is cut through:

```
cc_api_call_alert(vdbPtr=0x60B6C5D4, callID=0x2, prog_ind=0x8, sig_ind=0x1)
sess_appl: ev(6), cid(2), disp(0)
ssa: cid(2)st(1)oldst(0)cfid(-1)csiz(0)in(0)fDest(0)-cid2(1)st2(1)oldst2(0)
ccCallAlert (callID=0x1, prog_ind=0x8, sig_ind=0x1)
ccConferenceCreate (confID=0x60B6C150, callID1=0x1, callID2=0x2, tag=0x0)
cc_api_bridge_done (confID=0x1, srcIF=0x60B6C5D4, srcCallID=0x2, dstCallID=0x1,
disposition=0, tag=0x0)
cc_api_bridge_done (confID=0x1, srcIF=0x60BFB530, srcCallID=0x1, dstCallID=0x2,
disposition=0, tag=0x0)
cc_api_caps_ind (dstVdbPtr=0x60B6C5D4, dstCallId=0x2,srcCallId=0x1, caps={codec=0x7,
fax_rate=0x7F, vad=0x3})
cc_api_caps_ind (dstVdbPtr=0x60BFB530, dstCallId=0x1,srcCallId=0x2, caps={codec=0x4,
fax_rate=0x2, vad=0x2})
cc_api_caps_ack (dstVdbPtr=0x60BFB530, dstCallId=0x1,srcCallId=0x2, caps={codec=0x4,
fax_rate=0x2, vad=0x2})
cc_api_caps_ack (dstVdbPtr=0x60B6C5D4, dstCallId=0x2,srcCallId=0x1, caps={codec=0x4,
fax_rate=0x2, vad=0x2})
sess_appl: ev(17), cid(1), disp(0)
ssa: cid(1)st(3)oldst(0)cfid(1)csiz(0)in(1)fDest(0)-cid2(2)st2(3)oldst2(1)
```

The following output shows that the call is connected and voice is active:

```
cc_api_call_connected(vdbPtr=0x60B6C5D4, callID=0x2)
sess_appl: ev(7), cid(2), disp(0)
ssa: cid(2)st(4)oldst(1)cfid(1)csiz(0)in(0)fDest(0)-cid2(1)st2(4)oldst2(3)
ccCallConnect (callID=0x1)
```

The following output shows how the system processes voice statistics and monitors voice quality during the call:

```
ccapi_request_rt_packet_stats (requestorIF=0x60B6C5D4, requestorCID=0x2,
requestedCID=0x1, tag=0x60A7C598)
cc_api_request_rt_packet_stats_done (requestedIF=0x60BFB530, requestedCID=0x1,
tag=0x60A7A4C4)
ccapi_request_rt_packet_stats (requestorIF=0x60B6C5D4, requestorCID=0x2,
requestedCID=0x1, tag=0x60A7C598)
cc_api_request_rt_packet_stats_done (requestedIF=0x60BFB530, requestedCID=0x1,
tag=0x60C1FE54)
ccapi_request_rt_packet_stats (requestorIF=0x60B6C5D4, requestorCID=0x2,
requestedCID=0x1, tag=0x60A7C598)
cc_api_request_rt_packet_stats_done (requestedIF=0x60BFB530, requestedCID=0x1,
tag=0x60A7A5F4)
ccapi_request_rt_packet_stats (requestorIF=0x60B6C5D4, requestorCID=0x2,
requestedCID=0x1, tag=0x60A7C598)
cc_api_request_rt_packet_stats_done (requestedIF=0x60BFB530, requestedCID=0x1,
tag=0x60A7A6D8)
ccapi_request_rt_packet_stats (requestorIF=0x60B6C5D4, requestorCID=0x2,
requestedCID=0x1, tag=0x60A7C598)
cc_api_request_rt_packet_stats_done (requestedIF=0x60BFB530, requestedCID=0x1,
tag=0x60A7ACBC)
```

The following output shows that disconnection is indicated from the calling party. Call legs are torn down and disconnected:

```
cc_api_call_disconnected(vdbPtr=0x60BFB530, callID=0x1, cause=0x10)
sess_appl: ev(9), cid(1), disp(0)
ssa: cid(1)st(5)oldst(3)cfid(1)csize(0)in(1)fDest(0)-cid2(2)st2(5)oldst2(4)
ccConferenceDestroy (confID=0x1, tag=0x0)
cc_api_bridge_done (confID=0x1, srcIF=0x60B6C5D4, srcCallID=0x2, dstCallID=0x1,
disposition=0 tag=0x0)
cc_api_bridge_done (confID=0x1, srcIF=0x60BFB530, srcCallID=0x1, dstCallID=0x2,
disposition=0 tag=0x0)
sess_appl: ev(18), cid(1), disp(0)
ssa: cid(1)st(6)oldst(5)cfid(-1)csize(0)in(1)fDest(0)-cid2(2)st2(6)oldst2(4)
ccCallDisconnect (callID=0x1, cause=0x10 tag=0x0)
ccCallDisconnect (callID=0x2, cause=0x10 tag=0x0)
cc_api_call_disconnect_done(vdbPtr=0x60B6C5D4, callID=0x2, disp=0, tag=0x0)
sess_appl: ev(10), cid(2), disp(0)
ssa: cid(2)st(7)oldst(4)cfid(-1)csize(0)in(0)fDest(0)-cid2(1)st2(7)oldst2(6)
cc_api_call_disconnect_done(vdbPtr=0x60BFB530, callID=0x1, disp=0, tag=0x0)
sess_appl: ev(10), cid(1), disp(0)
ssa: cid(1)st(7)oldst(6)cfid(-1)csize(1)in(1)fDest(0)
```