

debug modem

Use the **debug modem** EXEC command to observe modem line activity on an access server. The **no** form of this command disables debugging output.

[no] debug modem

Sample Display

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

```
Router# debug modem

15:25:51: TTY4: DSR came up
15:25:51: tty4: Modem: IDLE->READY
15:25:51: TTY4: Autoselect started
15:27:51: TTY4: Autoselect failed
15:27:51: TTY4: Line reset
15:27:51: TTY4: Modem: READY->HANGUP
15:27:52: TTY4: dropping DTR, hanging up
15:27:52: tty4: Modem: HANGUP->IDLE
15:27:57: TTY4: restoring DTR
15:27:58: TTY4: DSR came up
```

The output shows when the modem line changes state.

debug modem csm

Use the **debug modem csm** EXEC command to debug the Call Switching Module (CSM), used to connect calls on the modem. The **no** form of this command disables debugging output.

[no] debug modem csm [*slot/port* | **group** *group-number*]

Syntax Description

slot/port (Optional) Slot and modem port number.

group *group-number* (Optional) Modem group.

Usage Guidelines

Use the **debug modem csm** command to troubleshoot call switching problems. With this command, you can trace the complete sequence of switching incoming and outgoing calls.

Sample Displays

The following is the sample output from the **debug modem csm** command. In this example, a call enters the modem (incoming) on slot 1 port 0.

```
Router(config)# service timestamps debug uptime
Router(config)# end
Router# debug modem csm

00:04:09: ccpri_ratetoteup bear rate is 10
00:04:09: CSM_MODEM_ALLOCATE: slot 1 and port 0 is allocated.
00:04:09: MODEM_REPORT(0001): DEV_INCALL at slot 1 and port 0
00:04:09: CSM_PROC_IDLE: CSM_EVENT_ISDN_CALL at slot 1, port 0
00:04:11: CSM_RING_INDICATION_PROC: RI is on
00:04:13: CSM_RING_INDICATION_PROC: RI is off
00:04:15: CSM_PROC_IC1_RING: CSM_EVENT_MODEM_OFFHOOK at slot 1, port 0
00:04:15: MODEM_REPORT(0001): DEV_CONNECTED at slot 1 and port 0
00:04:15: CSM_PROC_IC2_WAIT_FOR_CARRIER: CSM_EVENT_ISDN_CONNECTED at slot 1, port 0
```

The following is the sample output from the **debug modem csm** command when call is dialed from the modem into the network (outgoing) from slot 1 port 2:

```
Router# debug modem csm

atdt16665202
00:11:21: CSM_PROC_IDLE: CSM_EVENT_MODEM_OFFHOOK at slot 1, port 2
00:11:21: T1_MAIL_FROM_NEAT: DC_READY_RSP: mid = 1, slot = 0, unit = 0
00:11:21: CSM_PROC_OC1_REQUEST_DIGIT: CSM_EVENT_DIGIT_COLLECT_READY at slot 1, port 2
00:11:24: T1_MAIL_FROM_NEAT: DC_FIRST_DIGIT_RSP: mid = 1, slot = 0, unit = 0
00:11:24: CSM_PROC_OC2_COLLECT_1ST_DIGIT: CSM_EVENT_GET_1ST_DIGIT at slot 1, port 2
00:11:27: T1_MAIL_FROM_NEAT: DC_ALL_DIGIT_RSP: mid = 1, slot = 0, unit = 0
00:11:27: CSM_PROC_OC3_COLLECT_ALL_DIGIT: CSM_EVENT_GET_ALL_DIGITS (16665202) at slot 1, port 2
00:11:27: ccpri_ratetoteup bear rate is 10
00:11:27: MODEM_REPORT(A000): DEV_CALL_PROC at slot 1 and port 2
00:11:27: CSM_PROC_OC4_DIALING: CSM_EVENT_ISDN_BCHAN_ASSIGNED at slot 1, port 2
00:11:31: MODEM_REPORT(A000): DEV_CONNECTED at slot 1 and port 2
00:11:31: CSM_PROC_OC5_WAIT_FOR_CARRIER: CSM_EVENT_ISDN_CONNECTED at slot 1, port 2
CONNECT 19200/REL - MNP
```

The following is sample output from the **debug modem csm** command for an incoming call:

```

5300# debug modem csm
5300#1.19.36.7 2001
Trying 1.19.36.7, 2001 ... Open
atdt111222333444555666
*Apr 7 12:39:42.475: Mica Modem(1/0): Rcvd Dial String(111222333444555666)
*Apr 7 12:39:42.475: CSM_PROC_IDLE: CSM_EVENT_MODEM_OFFHOOK at slot 1, port 0
*Apr 7 12:39:42.479: CSM_RX_CAS_EVENT_FROM_NEAT:(A001): EVENT_CHANNEL_LOCK at slot 1
and port 0
*Apr 7 12:39:42.479: CSM_PROC_OC4_DIALING: CSM_EVENT_DSX0_BCHAN_ASSIGNED at slot 1,
port 0
*Apr 7 12:39:42.479: Mica Modem(1/0): Configure(0x1)
*Apr 7 12:39:42.479: Mica Modem(1/0): Configure(0x5)
*Apr 7 12:39:42.479: Mica Modem(1/0): Call Setup
*Apr 7 12:39:42.479: neat msg at slot 0: (1/0): Tx LOOP_CLOSURE (ABCD=1101)
*Apr 7 12:39:42.491: neat msg at slot 0: (0/0): Rx LOOP_CLOSURE (ABCD=1101)
*Apr 7 12:39:42.531: VDEV_ALLOCATE: slot 1 and port 3 is allocated.
*Apr 7 12:39:42.531: CSM_RX_CAS_EVENT_FROM_NEAT:(0004): EVENT_CALL_DIAL_IN at slot 1
and port 3
*Apr 7 12:39:42.531: CSM_PROC_IDLE: CSM_EVENT_DSX0_CALL at slot 1, port 3
*Apr 7 12:39:42.531: Mica Modem(1/3): Configure(0x0)
*Apr 7 12:39:42.531: Mica Modem(1/3): Configure(0x5)
*Apr 7 12:39:42.531: Mica Modem(1/3): Call Setup
*Apr 7 12:39:42.595: Mica Modem(1/0): State Transition to Call Setup
*Apr 7 12:39:42.655: Mica Modem(1/3): State Transition to Call Setup
*Apr 7 12:39:42.655: Mica Modem(1/3): Went offhook
*Apr 7 12:39:42.655: CSM_PROC_IC1_RING: CSM_EVENT_MODEM_OFFHOOK at slot 1, port 3
*Apr 7 12:39:42.671: neat msg at slot 0: (0/0): Tx LOOP_CLOSURE (ABCD=1101)
*Apr 7 12:39:42.691: neat msg at slot 0: (1/0): Rx LOOP_CLOSURE (ABCD=1101)
*Apr 7 12:39:42.731: CSM_RX_CAS_EVENT_FROM_NEAT:(A001): EVENT_START_TX_TONE at slot 1
and port 0
*Apr 7 12:39:42.731: CSM_PROC_OC4_DIALING: CSM_EVENT_DSX0_START_TX_TONE at slot 1,
port 0
*Apr 7 12:39:42.731: Mica Modem(1/0): Generate digits:called_party_num= len=1
*Apr 7 12:39:42.835: Mica Modem(1/3): Rcvd Digit detected(#)
*Apr 7 12:39:42.835: CSM_PROC_IC2_COLLECT_ADDR_INFO: CSM_EVENT_KP_DIGIT_COLLECTED
(DNIS=, ANI=) at slot 1, port 3
*Apr 7 12:39:42.855: neat msg at slot 0: (0/0): Tx LOOP_OPEN (ABCD=0101)
*Apr 7 12:39:42.871: neat msg at slot 0: (1/0): Rx LOOP_OPEN (ABCD=0101)
*Apr 7 12:39:42.899: Mica Modem(1/0): Rcvd Digits Generated
*Apr 7 12:39:42.911: CSM_RX_CAS_EVENT_FROM_NEAT:(A001): EVENT_END_TX_TONE at slot 1
and port 0
*Apr 7 12:39:42.911: CSM_PROC_OC4_DIALING: CSM_EVENT_DSX0_END_TX_TONE at slot 1, port
0
*Apr 7 12:39:42.911: Mica Modem(1/0): Generate digits:called_party_num=A len=1
*Apr 7 12:39:43.019: Mica Modem(1/0): Rcvd Digits Generated
*Apr 7 12:39:43.019: CSM_PROC_OC4_DIALING: CSM_EVENT_TONE_GENERATED at slot 1, port 0
*Apr 7 12:39:43.019: Mica Modem(1/3): Rcvd Digit detected(A)
*Apr 7 12:39:43.335: CSM_RX_CAS_EVENT_FROM_NEAT:(A001): EVENT_START_TX_TONE at slot 1
and port 0
*Apr 7 12:39:43.335: CSM_PROC_OC4_DIALING: CSM_EVENT_DSX0_START_TX_TONE at slot 1,
port 0
*Apr 7 12:39:43.335: Mica Modem(1/0): Generate
digits:called_party_num=111222333444555666 len=19
*Apr 7 12:39:43.439: Mica Modem(1/3): Rcvd Digit detected(1)
*Apr 7 12:39:43.559: Mica Modem(1/3): Rcvd Digit detected(1)
*Apr 7 12:39:43.619: Mica Modem(1/3): Rcvd Digit detected(1)
*Apr 7 12:39:43.743: Mica Modem(1/3): Rcvd Digit detected(2)
*Apr 7 12:39:43.859: Mica Modem(1/3): Rcvd Digit detected(2)
*Apr 7 12:39:43.919: Mica Modem(1/3): Rcvd Digit detected(2)
*Apr 7 12:39:44.043: Mica Modem(1/3): Rcvd Digit detected(3)
*Apr 7 12:39:44.163: Mica Modem(1/3): Rcvd Digit detected(3)
*Apr 7 12:39:44.223: Mica Modem(1/3): Rcvd Digit detected(3)
*Apr 7 12:39:44.339: Mica Modem(1/3): Rcvd Digit detected(4)

```

```
*Apr 7 12:39:44.459: Mica Modem(1/3): Rcvd Digit detected(4)
*Apr 7 12:39:44.523: Mica Modem(1/3): Rcvd Digit detected(4)
*Apr 7 12:39:44.639: Mica Modem(1/3): Rcvd Digit detected(5)
*Apr 7 12:39:44.763: Mica Modem(1/3): Rcvd Digit detected(5)
*Apr 7 12:39:44.883: Mica Modem(1/3): Rcvd Digit detected(5)
*Apr 7 12:39:44.943: Mica Modem(1/3): Rcvd Digit detected(6)
*Apr 7 12:39:45.063: Mica Modem(1/3): Rcvd Digit detected(6)
*Apr 7 12:39:45.183: Mica Modem(1/3): Rcvd Digit detected(6)
*Apr 7 12:39:45.243: Mica Modem(1/3): Rcvd Digit detected(B)
*Apr 7 12:39:45.243: CSM_PROC_IC2_COLLECT_ADDR_INFO: CSM_EVENT_DNIS_COLLECTED
(DNIS=111222333444555666, ANI=) at slot 1, port 3
*Apr 7 12:39:45.363: Mica Modem(1/0): Rcvd Digits Generated
*Apr 7 12:39:45.891: neat msg at slot 0: (0/0): Tx LOOP_CLOSURE (ABCD=1101)
*Apr 7 12:39:45.907: neat msg at slot 0: (1/0): Rx LOOP_CLOSURE (ABCD=1101)
*Apr 7 12:39:46.115: neat msg at slot 0: (0/0): Tx LOOP_OPEN (ABCD=0101)
*Apr 7 12:39:46.131: neat msg at slot 0: (1/0): Rx LOOP_OPEN (ABCD=0101)
*Apr 7 12:39:46.175: CSM_RX_CAS_EVENT_FROM_NEAT:(A001): EVENT_START_TX_TONE at slot 1
and port 0
*Apr 7 12:39:46.175: CSM_PROC_OC4_DIALING: CSM_EVENT_DSX0_START_TX_TONE at slot 1,
port 0
*Apr 7 12:39:46.175: Mica Modem(1/0): Generate digits:called_party_num= len=3
*Apr 7 12:39:46.267: Mica Modem(1/3): Rcvd Digit detected(#)
*Apr 7 12:39:46.387: Mica Modem(1/3): Rcvd Digit detected(A)
*Apr 7 12:39:46.447: Mica Modem(1/3): Rcvd Digit detected(B)
*Apr 7 12:39:46.447: CSM_PROC_IC2_COLLECT_ADDR_INFO: CSM_EVENT_ADDR_INFO_COLLECTED
(DNIS=111222333444555666, ANI=) at slot 1, port 3
*Apr 7 12:39:46.507: Mica Modem(1/0): Rcvd Digits Generated
*Apr 7 12:39:46.507: CSM_PROC_OC4_DIALING: CSM_EVENT_ADDR_INFO_COLLECTED at slot 1,
port 0
*Apr 7 12:39:47.127: CSM_RX_CAS_EVENT_FROM_NEAT:(0004): EVENT_CHANNEL_CONNECTED at
slot 1 and port 3
*Apr 7 12:39:47.127: CSM_PROC_IC4_WAIT_FOR_CARRIER: CSM_EVENT_DSX0_CONNECTED at slot
1, port 3
*Apr 7 12:39:47.127: Mica Modem(1/3): Link Initiate
*Apr 7 12:39:47.131: neat msg at slot 0: (0/0): Tx LOOP_CLOSURE (ABCD=1101)
*Apr 7 12:39:47.147: neat msg at slot 0: (1/0): Rx LOOP_CLOSURE (ABCD=1101)
*Apr 7 12:39:47.191: CSM_RX_CAS_EVENT_FROM_NEAT:(A001): EVENT_CHANNEL_CONNECTED at
slot 1 and port 0
*Apr 7 12:39:47.191: CSM_PROC_OC5_WAIT_FOR_CARRIER: CSM_EVENT_DSX0_CONNECTED at slot
1, port 0
*Apr 7 12:39:47.191: Mica Modem(1/0): Link Initiate
*Apr 7 12:39:47.227: Mica Modem(1/3): State Transition to Connect
*Apr 7 12:39:47.287: Mica Modem(1/0): State Transition to Connect
*Apr 7 12:39:49.103: Mica Modem(1/0): State Transition to Link
*Apr 7 12:39:52.103: Mica Modem(1/3): State Transition to Link
*Apr 7 12:40:00.927: Mica Modem(1/3): State Transition to Trainup
*Apr 7 12:40:00.991: Mica Modem(1/0): State Transition to Trainup
*Apr 7 12:40:02.615: Mica Modem(1/0): State Transition to EC Negotiating
*Apr 7 12:40:02.615: Mica Modem(1/3): State Transition to EC Negotiating
CONNECT 31200 /V.42/V.42bis
5300>
*Apr 7 12:40:05.983: Mica Modem(1/0): State Transition to Steady State
*Apr 7 12:40:05.983: Mica Modem(1/3): State Transition to Steady State+++
OK
ath
*Apr 7 12:40:09.167: Mica Modem(1/0): State Transition to Steady State Escape
*Apr 7 12:40:10.795: Mica Modem(1/0): State Transition to Terminating
*Apr 7 12:40:10.795: Mica Modem(1/3): State Transition to Terminating
*Apr 7 12:40:11.755: Mica Modem(1/3): State Transition to Idle
*Apr 7 12:40:11.755: Mica Modem(1/3): Went onhook
*Apr 7 12:40:11.755: CSM_PROC_IC5_OC6_CONNECTED: CSM_EVENT_MODEM_ONHOOK at slot 1,
port 3
*Apr 7 12:40:11.755: VDEV_DEALLOCATE: slot 1 and port 3 is deallocated
*Apr 7 12:40:11.759: neat msg at slot 0: (0/0): Tx LOOP_OPEN (ABCD=0101)
*Apr 7 12:40:11.767: neat msg at slot 0: (1/0): Rx LOOP_OPEN (ABCD=0101)
```

```

*Apr 7 12:40:12.087: neat msg at slot 0: (1/0): Tx LOOP_OPEN (ABCD=0101)
*Apr 7 12:40:12.091: neat msg at slot 0: (0/0): Rx LOOP_OPEN (ABCD=0101)
*Apr 7 12:40:12.111: CSM_RX_CAS_EVENT_FROM_NEAT:(A001): EVENT_CALL_IDLE at slot 1 and
port 0
*Apr 7 12:40:12.111: CSM_PROC_IC5_OC6_CONNECTED: CSM_EVENT_DSX0_DISCONNECTED at slot
1, port 0
*Apr 7 12:40:12.111: Mica Modem(1/0): Link Terminate(0x6)
*Apr 7 12:40:12.779: Mica Modem(1/3): State Transition to Terminating
*Apr 7 12:40:12.839: Mica Modem(1/3): State Transition to Idle
*Apr 7 12:40:13.495: Mica Modem(1/0): State Transition to Idle
*Apr 7 12:40:13.495: Mica Modem(1/0): Went onhook
*Apr 7 12:40:13.495: CSM_PROC_IC6_OC8_DISCONNECTING: CSM_EVENT_MODEM_ONHOOK at slot 1,
port 0
*Apr 7 12:40:13.495: VDEV_DEALLOCATE: slot 1 and port 0 is deallocated
5300#disc
Closing connection to 1.19.36.7 [confirm]
5300#
*Apr 7 12:40:18.783: Mica Modem(1/0): State Transition to Terminating
*Apr 7 12:40:18.843: Mica Modem(1/0): State Transition to Idle
5300#

```

The MICA modem goes through the following internal link states when the call comes in: 1. Call Setup, 2. Off Hook, 3. Connect, 4. Link, 5. Trainup, 6. EC Negotiation, 7. Steady State

The following describes the CSM activity for an incoming call:

When a voice call comes in, CSM is informed of the incoming call. This allocates the modem and sends the Call Setup message to the MICA modem. The Call_Proc message is sent through D channel. The modem sends an offhook message to CSM by sending the state change to Call Setup. The D channel then sends a CONNECT message. When the CONNECT_ACK message is received, the Link initiate message is sent to the MICA modem and it negotiates the connection with remote modem. In these debug examples, a modem on slot 1 and port 13 is allocated. It goes through its internal states before it is in Steady State and answers the call.

```

5300# debug modem csm
Modem Management Call Switching Module debugging is on
*May 13 15:01:00.609: MODEM_REPORT:dchan_idb=0x60D437F8, call_id=0xE, ces=0x1
bchan=0x12, event=0x1, cause=0x0
*May 13 15:01:00.609: VDEV_ALLOCATE: slot 1 and port 13 is allocated.
*May 13 15:01:00.609: MODEM_REPORT(000E): DEV_INCALL at slot 1 and port 13
*May 13 15:01:00.609: CSM_PROC_IDLE: CSM_EVENT_ISDN_CALL at slot 1, port 13
*May 13 15:01:00.609: Mica Modem(1/13): Configure(0x0)
*May 13 15:01:00.609: Mica Modem(1/13): Configure(0x0)
*May 13 15:01:00.609: Mica Modem(1/13): Configure(0x6)
*May 13 15:01:00.609: Mica Modem(1/13): Call Setup
*May 13 15:01:00.661: Mica Modem(1/13): State Transition to Call Setup
*May 13 15:01:00.661: Mica Modem(1/13): Went offhook
*May 13 15:01:00.661: CSM_PROC_IC1_RING: CSM_EVENT_MODEM_OFFHOOK at slot 1, port 13
*May 13 15:01:00.661: MODEM_REPORT:dchan_idb=0x60D437F8, call_id=0xE, ces=0x1
bchan=0x12, event=0x4, cause=0x0
*May 13 15:01:00.661: MODEM_REPORT(000E): DEV_CONNECTED at slot 1 and port 13
*May 13 15:01:00.665: CSM_PROC_IC3_WAIT_FOR_CARRIER:
CSM_EVENT_ISDN_CONNECTED at slot 1, port 13
*May 13 15:01:00.665: Mica Modem(1/13): Link Initiate
*May 13 15:01:00.693: Mica Modem(1/13): State Transition to Connect
*May 13 15:01:01.109: Mica Modem(1/13): State Transition to Link
*May 13 15:01:09.433: Mica Modem(1/13): State Transition to Trainup
*May 13 15:01:11.541: Mica Modem(1/13): State Transition to EC Negotiating
*May 13 15:01:12.501: Mica Modem(1/13): State Transition to Steady State

```

The following describes the status of CSM when a call is connected:

The show modem csm x/y is similar to AS5200. For an active incoming analog call, the modem_status and csm_status should be VDEV_STATUS_ACTIVE_CALL and CSM_IC4_CONNECTED, respectively.

```
5300# show modem csm 1/13
MODEM_INFO: slot 1, port 13, unit 0, modem_mask=0x0000, modem_port_offset=0
tty_hwidb=0x60D0BCE0, modem_tty=0x60B6FE7C, oobp_info=0x00000000,
modem_pool=0x60ADC998
modem_status(0x0002):VDEV_STATUS_ACTIVE_CALL.
csm_state(0x0204)=CSM_IC4_CONNECTED, csm_event_proc=0x600C6968, current
call thru PRI line
invalid_event_count=0, wdt_timeout_count=0
wdt_timestamp_started is not activated
wait_for_dialing:False, wait_for_bchan:False
pri_chnl=TDM_PRI_STREAM(s0, u0, c18), modem_chnl=TDM_MODEM_STREAM(s1, c13)
dchan_idb_start_index=0, dchan_idb_index=0, call_id=0x000E, bchan_num=18
csm_event=CSM_EVENT_ISDN_CONNECTED, cause=0x0000
ring_indicator=0, oh_state=0, oh_int_enable=0, modem_reset_reg=0
ring_no_answer=0, ic_failure=0, ic_complete=1
dial_failure=0, oc_failure=0, oc_complete=0
oc_busy=0, oc_no_dial_tone=0, oc_dial_timeout=0
remote_link_disc=0, stat_busyout=0, stat_modem_reset=0
oobp_failure=0
call_duration_started=1d02h, call_duration_ended=00:00:00,
total_call_duration=00:00:00
The calling party phone number = 4085552400
The called party phone number = 4085551400
total_free_rbs_timeslot = 0, total_busy_rbs_timeslot = 0,
total_dynamic_busy_rbs_timeslot = 0, total_static_busy_rbs_timeslot = 0,
min_free_modem_threshold = 6
```

The following describes the CSM activity for an outgoing call:

For MICA modems, the dial tone is not required to initiate an outbound call. Unlike the AS5200, the digit collection step is not required. The dialed digit string is sent to CSM in the outgoing request to CSM. CSM signals the D-channel to generate an outbound voice-call, and the B-channel assigned is connected to modem and CSM.

The modem is ordered to connect to the remote side with a CONNECT message, and by sending a link initiate message, then the modem starts to train.

```
5300# debug modem csm
Modem Management Call Switching Module debugging is on
5300# debug isdn q931
ISDN Q931 packets debugging is on
*May 15 12:48:42.377: Mica Modem(1/0): Rcvd Dial String(5552400)
*May 15 12:48:42.377: CSM_PROC_IDLE: CSM_EVENT_MODEM_OFFHOOK at slot 1, port 0
*May 15 12:48:42.377: CSM_PROC_OC3_COLLECT_ALL_DIGIT:
CSM_EVENT_GET_ALL_DIGITS at slot 1, port 0
*May 15 12:48:42.377: CSM_PROC_OC3_COLLECT_ALL_DIGIT: called party num:
(5552400) at slot 1, port 0
*May 15 12:48:42.381: process_pri_call making a voice_call.
*May 15 12:48:42.381: ISDN Se0:23: TX -> SETUP pd = 8 callref = 0x0011
*May 15 12:48:42.381: Bearer Capability i = 0x8090A2
*May 15 12:48:42.381: Channel ID i = 0xE1808397
*May 15 12:48:42.381: Called Party Number i = 0xA1, '5552400'
*May 15 12:48:42.429: ISDN Se0:23: RX <- CALL_PROC pd = 8 callref = 0x8011
*May 15 12:48:42.429: Channel ID i = 0xA98397
*May 15 12:48:42.429: MODEM_REPORT:dchan_idb=0x60D437F8, call_id=0xA011, ces=0x1
bchan=0x16, event=0x3, cause=0x0
*May 15 12:48:42.429: MODEM_REPORT(A011): DEV_CALL_PROC at slot 1 and port 0
*May 15 12:48:42.429: CSM_PROC_OC4_DIALING: CSM_EVENT_ISDN_BCHAN_ASSIGNED
```

```
at slot 1, port 0
*May 15 12:48:42.429: Mica Modem(1/0): Configure(0x1)
*May 15 12:48:42.429: Mica Modem(1/0): Configure(0x0)
*May 15 12:48:42.429: Mica Modem(1/0): Configure(0x6)
*May 15 12:48:42.429: Mica Modem(1/0): Call Setup
*May 15 12:48:42.489: Mica Modem(1/0): State Transition to Call Setup
*May 15 12:48:42.589: ISDN Se0:23: RX <- ALERTING pd = 8 callref = 0x8011
*May 15 12:48:43.337: ISDN Se0:23: RX <- CONNECT pd = 8 callref = 0x8011
*May 15 12:48:43.341: MODEM_REPORT:dchan_idb=0x60D437F8, call_id=0xA011, ces=0x1
    bchan=0x16, event=0x4, cause=0x0
*May 15 12:48:43.341: MODEM_REPORT(A011): DEV_CONNECTED at slot 1 and port 0
*May 15 12:48:43.341: CSM_PROC_OC5_WAIT_FOR_CARRIER:
CSM_EVENT_ISDN_CONNECTED at slot 1, port 0
*May 15 12:48:43.341: Mica Modem(1/0): Link Initiate
*May 15 12:48:43.341: ISDN Se0:23: TX -> CONNECT_ACK pd = 8 callref = 0x0011
*May 15 12:48:43.385: Mica Modem(1/0): State Transition to Connect
*May 15 12:48:43.849: Mica Modem(1/0): State Transition to Link
*May 15 12:48:52.665: Mica Modem(1/0): State Transition to Trainup
*May 15 12:48:54.661: Mica Modem(1/0): State Transition to EC Negotiating
*May 15 12:48:54.917: Mica Modem(1/0): State Transition to Steady State
```

Related Commands

debug modem oob
debug modem trace

debug modem oob

Use the **debug modem oob** EXEC command to debug the out-of-band port used to poll modem events on the modem. The **no** form of this command disables debugging output.

```
[no] debug modem oob [slot/modem-port | group group-number]
```

Syntax Description

| | |
|----------------------------------|--|
| <i>slot/modem-port</i> | (Optional) Slot and modem port number. |
| group <i>group-number</i> | (Optional) Modem group. |

Usage Guidelines



Caution Entering the **debug modem oob** command without specifying a slot and modem number debugs *all* out-of-band ports, which generates a significant amount of information.

The message types and sequence numbers that appear in the debug output are initiated by the Modem Out-of-Band (OOB) Protocol and used by service personnel for debugging purposes.

Sample Display

The following is sample output from the **debug modem oob** command. This example debugs the out-of-band port on modem 2/0, which creates modem startup messages between the network management software and the modem.

```
Router# debug modem oob 2/0

MODEM(2/0): One message sent --Message type:3, Sequence number:0
MODEM(2/0): Modem DC session data reply
MODEM(2/0): One message sent --Message type:83, Sequence number:1
MODEM(2/0): DC session event =
MODEM(2/0): One message sent --Message type:82, Sequence number:2
MODEM(2/0): No status changes since last polled
MODEM(2/0): One message sent --Message type:3, Sequence number:3
MODEM(2/0): Modem DC session data reply
MODEM(2/0): One message sent --Message type:83, Sequence number:4
```

Related Commands

debug modem csm
debug modem trace

debug modem trace

Use the **debug modem trace** EXEC command to debug a call trace on the modem to determine why calls are terminated. The **no** form of this command disables debugging output.

[no] debug modem trace [normal | abnormal | all] [slot/modem-port | group group-number]

Syntax Description

| | |
|----------------------------------|--|
| normal | (Optional) Uploads the call trace to the syslog server on normal call termination (for example, a local user hangup or a remote user hangup). |
| abnormal | (Optional) Uploads the call trace to the syslog server on abnormal call termination (for example, any call termination other than normal termination, such as a lost carrier or a watchdog timeout). |
| all | (Optional) Uploads the call trace on all call terminations including normal and abnormal call termination. |
| <i>slot/modem-port</i> | (Optional) Slot and modem port number. |
| group <i>group-number</i> | (Optional) Modem group. |

Usage Guidelines

The **debug modem trace** command applies only to manageable modems. For additional information, use the **show modem** command.

Sample Display

The following is sample output from the **debug modem trace abnormal** command:

```
Router# debug modem trace abnormal 1/14

Modem 1/14 Abnormal End of Connection Trace. Caller 123-4567
  Start-up Response: AS5200 Modem, Firmware 1.0
  Control Reply: 0x7C01
  DC session response: brasil firmware 1.0
  RS232 event:
  DSR=On, DCD=On, RI=Off, TST=Off
  changes: RTS=No change, DTR=No change, CTS=No change
  changes: DSR=No change, DCD=No change, RI=No change, TST=No change
  Modem State event: Connected
  Connection event: Speed = 19200, Modulation = VFC
  Direction = Originate, Protocol = reliable/LAPM, Compression = V42bis
  DTR event: DTR On
  Modem Activity event: Data Active
  Modem Analog signal event: TX = -10, RX = -24, Signal to noise = -32
  End connection event: Duration = 10:34-11:43,
  Number of xmit char = 67, Number of rcvd char = 88, Reason: Watchdog Time-out.
```

Related Commands

debug modem csm
debug modem oob

debug mpoa client

Use the **debug mpoa client** EXEC command to display MPC debug information. The **no** form of this command disables debugging output.

```
[no] debug mpoa client {all | data | egress | general | ingress | keep-alives | platform-specific}
[name mpc-name]
```

Syntax Description

| | |
|-----------------------------|--|
| all | (Optional) Keyword that shows debugging information for all MPC activity. |
| data | (Optional) Keyword that shows debugging information for data plane activity only. This option applies only to routers. |
| egress | (Optional) Keyword that shows debugging information for egress functionality only. |
| general | (Optional) Keyword that shows general debugging information only. |
| ingress | (Optional) Keyword that shows debugging information for ingress functionality only. |
| keep-alives | (Optional) Keyword that shows debugging information for keepalive activity only. |
| platform-specific | (Optional) Keyword that show debugging information for specific platforms only. This option applies only to the Catalyst 5000 series ATM module. |
| name <i>mpc-name</i> | (Optional) Keyword that specifies the name of the MPC with the specified name. |

Default

The default is debugging turned on for all MPCs.

Sample Display

The following shows how to turn on debugging for the MPC ip_mpc:

```
ATM#debug mpoa client all name ip_mpc
```

debug mpoa server

Use the **debug mpoa server** EXEC command to display information about the MPOA server. This command optionally limits the output only to the specified MPS. The **no** form of this command disables debugging output.

[no] debug mpoa server [name *mps-name*]

Syntax Description

name *mps-name* (Optional) Specifies the name of a MPOA server.

Sample Display

The following turns on debugging only for the MPS named ip_mps:

```
Router# debug mpoa server name ip_mps
```

debug ncia circuit

Use the **debug ncia circuit** EXEC command to display circuit-related information between the native client interface architecture (NCIA) server and client. The **no** form of this command disables debugging output.

[no] debug ncia circuit [error | event | flow-control | state]

Syntax Description

| | |
|---------------------|--|
| error | (Optional) Displays the error situation for each circuit. |
| event | (Optional) Displays the packets received and transmitted for each circuit. |
| flow-control | (Optional) Displays the flow control information for each circuit. |
| state | (Optional) Displays the state changes for each circuit. |

Usage Guidelines

NCIA is an architecture developed by Cisco for accessing SNA applications. This architecture allows native SNA interfaces on hosts and clients to access TCP/IP backbones.

You cannot enable debugging output for a particular client or particular circuit.



Caution Do not enable the **debug ncia circuit** command during normal operation because this command generates a large amount of output messages and could slow down the router.

Sample Displays

The following is sample output from the **debug ncia circuit error** command. In this example, the possible errors are displayed. The first error message indicates that the router is out of memory. The second message indicates that the router has an invalid circuit control block. The third message indicates that the router is out of memory. The remaining messages identify errors related to the finite state machine.

```
Router# debug ncia circuit error

NCIA: ncia_circuit_create memory allocation fail
NCIA: ncia_send_ndlc: invalid circuit control block
NCIA: send_ndlc: fail to get buffer for ndlc primitive xxx
NCIA: ncia circuit fsm: Invalid input
NCIA: ncia circuit fsm: Illegal state
NCIA: ncia circuit fsm: Illegal input
NCIA: ncia circuit fsm: Unexpected input
NCIA: ncia circuit fsm: Unknown error rtn code
```

The following is sample output from the **debug ncia circuit event** command. In this example, a session start-up sequence is displayed.

```
Router# debug ncia circuit event

NCIA(IN): Ver_Id: 0x81, MsgType: NDLC_START_DL, Len: 24, tmac: 4000.1060.1000,
         tsap: 4, csap 8, oid: 8A91E8, tid 0, lfs 16, ws 1
NCIA: create circuit: saddr 4000.1060.1000, ssap 4, daddr 4000.3000.0003, dsap 8 sid:
         8B09A8
NCIA: send NDLC_DL_STARTED to client 10.2.20.3 for ckt: 8B09A8
NCIA(OUT): Ver_Id: 0x81, MsgType: NDLC_DL_STARTED, Len: 2,4 tmac: 4000.1060.1000,
         tsap: 4, csap 8, oid: 8A91E8, tid 8B09A8, lfs 16, ws 1
NCIA(IN): Ver_Id: 0x81, MsgType: NDLC_XID_FRAME, Len: 12, sid: 8B09A8, FC 0x81
NCIA: send NDLC_XID_FRAME to client 10.2.20.3 for ckt: 8B09A8
NCIA(OUT): Ver_Id: 0x81, MsgType: NDLC_XID_FRAME, Len: 12, sid: 8A91E8, FC 0xC1
NCIA(IN): Ver_Id: 0x81, MsgType: NDLC_XID_FRAME, Len: 18, sid: 8B09A8, FC 0xC1
NCIA: send NDLC_CONTACT_STN to client 10.2.20.3 for ckt: 8B09A8
NCIA(OUT): Ver_Id: 0x81, MsgType: NDLC_CONTACT_STN, Len: 12, sid: 8A91E8, FC 0xC1
NCIA(IN): Ver_Id: 0x81, MsgType: NDLC_STN_CONTACTED, Len: 12, sid: 8B09A8, FC 0xC1
NCIA: send NDLC_INFO_FRAME to client 10.2.20.3 for ckt: 8B09A8
NCIA(OUT): Ver_Id: 0x81, MsgType: NDLC_INFO_FRAME, Len: 30, sid: 8A91E8, FC 0xC1
```

Table 92 describes the significant fields in the output.

Table 92 Debug NCIA Circuit Event Field Descriptions

| Field | Description |
|---------|-------------------------------|
| IN | Incoming message from client. |
| OUT | Outgoing message to client. |
| Ver_Id | NDLC version ID. |
| MsgType | NDLC message type. |
| Len | NDLC message length. |
| tmac | Target MAC. |
| tsap | Target SAP. |
| csap | Client SAP. |
| oid | Origin ID. |
| tid | Target ID |
| lfs | Largest frame size flag. |
| ws | Window size. |
| saddr | Source MAC address. |
| ssap | Source SAP. |
| daddr | Destination MAC address. |
| dsap | Destination SAP. |
| sid | Session ID. |
| FC | Flow control flag. |

In the following messages, an NDLC_START_DL messages is received from a client. to start a data link session:

```
NCIA(IN): Ver_Id: 0x81, MsgType: NDLC_START_DL, Len: 24, tmac: 4000.1060.1000,
         tsap: 4, csap 8, oid: 8A91E8, tid 0, lfs 16, ws 1
NCIA: create circuit: saddr 4000.1060.1000, ssap 4, daddr 4000.3000.0003, dsap 8 sid:
      8B09A8
```

The next two messages indicate that an NDLC_DL_STARTED message is sent to a client. The server informs the client that a data link session is started.

```
NCIA: send NDLC_DL_STARTED to client 10.2.20.3 for ckt: 8B09A8
NCIA(OUT): Ver_Id: 0x81, MsgType: NDLC_DL_STARTED, Len: 2,4 tmac: 4000.1060.1000,
          tsap: 4, csap 8, oid: 8A91E8, tid 8B09A8, lfs 16, ws 1
```

In the following two messages, an NDLC_XID_FRAME message is received from a client, and the client starts an XID exchange:

```
NCIA(IN): Ver_Id: 0x81, MsgType: NDLC_XID_FRAME, Len: 12, sid: 8B09A8, FC 0x81
NCIA: send NDLC_XID_FRAME to client 10.2.20.3 for ckt: 8B09A8
```

In the following two messages, an NDLC_XID_FRAME message is sent from a client, and an DLC_XID_FRAME message is received from a client:

```
NCIA(OUT): Ver_Id: 0x81, MsgType: NDLC_XID_FRAME, Len: 12, sid: 8A91E8, FC 0xC1
NCIA(IN): Ver_Id: 0x81, MsgType: NDLC_XID_FRAME, Len: 18, sid: 8B09A8, FC 0xC1
```

The next two messages show that an NDLC_CONTACT_STN message is sent to a client:

```
NCIA: send NDLC_CONTACT_STN to client 10.2.20.3 for ckt: 8B09A8
NCIA(OUT): Ver_Id: 0x81, MsgType: NDLC_CONTACT_STN, Len: 12, sid: 8A91E8, FC 0xC1
```

In the following message, an NDLC_STN_CONTACTED message is received from a client. The client informs server that the station has been contacted.

```
NCIA(IN): Ver_Id: 0x81, MsgType: NDLC_STN_CONTACTED, Len: 12, sid: 8B09A8, FC 0xC1
```

In the last two messages, an NDLC_INFO_FRAME is sent to a client, and the server sends data to the client:

```
NCIA: send NDLC_INFO_FRAME to client 10.2.20.3 for ckt: 8B09A8
NCIA(OUT): Ver_Id: 0x81, MsgType: NDLC_INFO_FRAME, Len: 30, sid: 8A91E8, FC 0xC1
```

The following is sample output from the **debug ncia circuit flow-control** command. In this example, the flow control in a session start-up sequence is displayed.

```
Router# debug ncia circuit flow-control

NCIA: no flow control in NDLC_DL_STARTED frame
NCIA: receive Increment Window Op for circuit 8ADE00
NCIA: ncia_flow_control_in FC 0x81, IW 1 GP 2 CW 2, Client IW 1 GP 0 CW 1
NCIA: grant client more packet by sending Repeat Window Op
NCIA: ncia_flow_control_out FC: 0xC1, IW 1 GP 2 CW 2, Client IW 1 GP 2 CW 2
NCIA: receive FCA for circuit 8ADE00
NCIA: receive Increment Window Op for circuit 8ADE00
NCIA: ncia_flow_control_in FC 0xC1, IW 1 GP 5 CW 3, Client IW 1 GP 2 CW 2
NCIA: grant client more packet by sending Repeat Window Op
NCIA: ncia_flow_control_out FC: 0xC1, IW 1 GP 5 CW 3, Client IW 1 GP 5 CW 3
NCIA: receive FCA for circuit 8ADE00
NCIA: receive Increment Window Op for circuit 8ADE00
NCIA: ncia_flow_control_in FC 0xC1, IW 1 GP 9 CW 4, Client IW 1 GP 5 CW 3
NCIA: grant client more packet by sending Repeat Window Op
NCIA: ncia_flow_control_out FC: 0xC1, IW 1 GP 8 CW 4, Client IW 1 GP 9 CW 4
```

```
NCIA: reduce ClientGrantPacket by 1 (Granted: 8)
NCIA: receive FCA for circuit 8ADE00
NCIA: receive Increment Window Op for circuit 8ADE00
```

Table 93 describes the significant fields in the output.

Table 93 Debug NCIA Circuit Flow-Control Field Descriptions

| Field | Description |
|-------|------------------------|
| IW | Initial window size. |
| GP | Granted packet number. |
| CW | Current window size. |

The following is sample output from the **debug ncia circuit state** command. In this example, a session start-up sequence is displayed.

```
Router# debug ncia circuit state

NCIA: pre-server fsm: event CONN_OPENED
NCIA: pre-server fsm: event NDLC_PRIMITIVES
NCIA: server event: WAN - STDL state: CLSOED
NCIA: ncia server fsm action 32
NCIA: circuit state: CLOSED -> START_DL_RCVD
NCIA: server event: DLU - TestStn.Rsp state: START_DL_RCVD
NCIA: ncia server fsm action 17
NCIA: circuit state: START_DL_RCVD -> DL_STARTED_SND
NCIA: pre-server fsm: event NDLC_PRIMITIVES
NCIA: server event: WAN - XID state: DL_STARTED_SND
NCIA: ncia server fsm action 33
NCIA: circuit state: DL_STARTED_SND -> DL_STARTED_SND
NCIA: server event: DLU - ReqOpnStn.Reg state: DL_STARTED_SND
NCIA: ncia server fsm action 33
NCIA: circuit state: DL_STARTED_SND -> OPENED
NCIA: server event: DLU - Id.Rsp state: OPENED
NCIA: ncia server fsm action 11
NCIA: circuit state: OPENED -> OPENED
NCIA: pre-server fsm: event NDLC_PRIMITIVES
NCIA: server event: WAN - XID state: OPENED
NCIA: ncia server fsm action 33
NCIA: circuit state: OPENED -> OPENED
NCIA: server event: DLU - Connect.Reg state: OPENED
NCIA: ncia server fsm action 6
NCIA: circuit state: OPENED -> CONNECT_PENDING
NCIA: pre-server fsm: event NDLC_PRIMITIVES
NCIA: server event: WAN - CONR state: CONNECT_PENDING
NCIA: ncia server fsm action 33 --> CLS_CONNECT_CNF sets NciaClsBusy
NCIA: circuit state: CONNECT_PENDING -> CONNECTED
NCIA: server event: DLU - Flow.Reg (START) state: CONNECTED
NCIA: ncia server fsm action 25 --> unset NciaClsBusy
NCIA: circuit state: CONNECTED -> CONNECTED
NCIA: server event: DLU - Data.Rsp state: CONNECTED
NCIA: ncia server fsm action 8
NCIA: circuit state: CONNECTED -> CONNECTED
```

Table 94 describes the significant fields in the output.

Table 94 **Debug NCIA Circuit State Field Descriptions**

| Field | Description |
|--------------|--|
| WAN | Event from WAN (client). |
| DLU | Event from upstream module—dependent logical unit (DLU). |
| ADMIN | Administrative event. |
| TIMER | Timer event. |

Related Commands

debug dlsw
debug ncia client
debug ncia server

debug ncia client

Use the **debug ncia client** EXEC command to display debug information for all native client interface architecture (NCIA) client processing that occurs in the router. The **no** form of this command disables debugging output.

```
[no] debug ncia client [ip-address | error [ip-address] | event [ip-address] | message [ip-address]]
```

Syntax Description

| | |
|-------------------|---|
| <i>ip-address</i> | (Optional) Remote client IP address. |
| error | (Optional) Triggers the recording of messages only when errors occur. The current state and event of a NCIA client are normally included in the message. If you do not specify an IP address, the error messages are logged for all active clients. |
| event | (Optional) Triggers the recording of messages that describe the current state and event—and sometimes the action that just completed—for the NCIA client. If you do not specify an IP address, the messages are logged for all active clients. |
| message | (Optional) Triggers the recording of messages that contain up to the first 32 bytes of data in a TCP packet sent to or received from an NCIA client. If you do not specify an IP address, the messages are logged for all active clients. |

Usage Guidelines

NCIA is an architecture developed by Cisco for accessing SNA applications. This architecture allows native SNA interfaces on hosts and clients to access TCP/IP backbones.

Use the **debug ncia client event** command to determine the sequences of activities that occur while a NCIA client is in different processing states.

Use the **debug ncia client error** command to see only certain error conditions that occur.

Use the **debug ncia client message** command to see only the first 32 bytes of data in a TCP packet sent to or received from an NCIA client.

The **debug ncia client** command can be used in conjunction with the **debug ncia server** and **debug ncia circuit** commands to get a complete picture of NCIA activity.

Sample Display

The following is sample output from the **debug ncia circuit** command. Following the example is a description of each sample output message.

```
Router# debug ncia client

NCIA: Passive open 10.2.20.123(1088) -> 1973
NCIA: index for client hash queue is 27
NCIA: number of element in client hash queue 27 is 1
NCIA: event PASSIVE_OPEN, state NCIA_CLOSED for client 10.2.20.123
NCIA: Rcvd msg type NDLC_CAP_XCHG in tcp packet for client 10.2.20.123
NCIA: First 17 byte of data rcvd: 81120011000000000000400050104080C
NCIA: Sent msg type NDLC_CAP_XCHG in tcp packet to client 10.2.20.123
NCIA: First 17 byte of data sent: 811200111000000010000400050104080C
NCIA: event CAP_CMD_RCVD, state NCIA_CAP_WAIT, for client 10.2.20.123, cap xchg cmd
sent
NCIA: Rcvd msg type NDLC_CAP_XCHG in tcp packet for client 10.2.20.123
NCIA: First 17 byte of data rcvd: 811200111000000010000000050104080C
NCIA: event CAP_RSP_RCVD, state NCIA_CAP_NEG for client 10.2.20.123

NCIA: Rcvd msg type NDLC_PEER_TEST_REQ in tcp packet for client 10.2.20.123
NCIA: First 4 byte of data rcvd: 811D0004
NCIA: event KEEPALIVE_RCVD, state NCIA_OPENED for client 10.2.20.123
NCIA: Sent msg type NDLC_PEER_TEST_RSP in tcp packet to client 10.2.20.123
NCIA: First 4 byte of data sent: 811E0004IA

NCIA: event TIME_OUT, state NCIA_OPENED, for client 10.2.20.123, keepalive_count = 0
NCIA: Sent msg type NDLC_PEER_TEST_REQ, in tcp packet to client 10.2.20.123
NCIA: First 4 byte of data sent: 811D0004
NCIA: Rcvd msg type NDLC_PEER_TEST_RSP in tcp packet for client 10.2.20.123
NCIA: First 4 byte of data rcvd: 811E0004
NCIA: event KEEPALIVE_RSP_RCVD, state NCIA_OPENED for client 10.2.20.123

NCIA: Error, event PASIVE_OPEN, state NCIA_OPENED, for client 10.2.20.123, should not
have occurred.
NCIA: Error, active_open for pre_client_fsm while client 10.2.20.123 is active or not
configured, registered.
```

Messages in lines 1 through 12 show the events that occur when a client connects to the router (the NCIA server). These messages show a passive_open process.

Messages in lines 13 to 17 show the events that occur when a TIME_OUT event is detected by a client PC workstation. The workstation sends an NDLC_PEER_TEST_REQ message to the NCIA server, and the router responds with an NDLC_PEER_TEST_RSP message.

Messages in lines 18 to 23 show the events that occur when a TIME_OUT event is detected by the router (the NCIA server). The router sends an NDLC_PEER_TEST_REQ message to the client PC workstation, and the PC responds with an NDLC_PEER_TEST_RSP message.

When you use the **debug ncia client message** command, the messages shown on lines 6, 8, 11, 14, 17, 20, and 22 are output in addition to other messages not shown in this example.

When you use the **debug ncia client error** command, the messages shown on lines 24 and 25 are output in addition to other messages not shown in this example.

Related Commands

```
debug ncia circuit
debug ncia server
```

debug ncia server

Use the **debug ncia server** EXEC command to display debug information for the native client interface architecture (NCIA) server and its upstream software modules. The **no** form of this command disables debugging output.

[no] debug ncia server

Usage Guidelines

NCIA is an architecture developed by Cisco for accessing SNA applications. This architecture allows native SNA interfaces on hosts and clients to access TCP/IP backbones.

The **debug ncia server** command displays all Cisco Link Services (CLS) messages between the NCIA server and its upstream modules, such as data-link switching (DLSw) and downstream physical units (DSPUs). Use this command when a problem exists between the NCIA server and other software modules within the router.

You cannot enable debugging output for a particular client or particular circuit.

Sample Display

The following is sample output from the **debug ncia server** command. In this example, a session start-up sequence is displayed. Following the example is a description of each group of sample output messages.

```
Router# debug ncia server

NCIA: send CLS_TEST_STN_IND to DLU
NCIA: Receive TestStn.Rsp
NCIA: send CLS_ID_STN_IND to DLU
NCIA: Receive ReqOpnStn.Req
NCIA: send CLS_REQ_OPNSTN_CNF to DLU
NCIA: Receive Id.Rsp
NCIA: send CLS_ID_IND to DLU
NCIA: Receive Connect.Req
NCIA: send CLS_CONNECT_CNF to DLU
NCIA: Receive Flow.Req
NCIA: Receive Data.Req
NCIA: send CLS_DATA_IND to DLU
NCIA: send CLS_DISC_IND to DLU
NCIA: Receive Disconnect.Rsp
```

In the following messages, the client is sending a test message to the host and the test message is received by the host:

```
NCIA: send CLS_TEST_STN_IND to DLU
NCIA: Receive TestStn.Rsp
```

In the next message, the server is sending an XID message to the host:

```
NCIA: send CLS_ID_STN_IND to DLU
```

In the next two messages, the host opens the station and the server responds:

```
NCIA: Receive ReqOpnStn.Req
NCIA: send CLS_REQ_OPNSTN_CNF to DLU
```

In the following two messages, the client is performing an XID exchange with the host:

```
NCIA: Receive Id.Rsp
NCIA: send CLS_ID_IND to DLU
```

In the next group of messages, the host attempts to establish a session with the client:

```
NCIA: Receive Connect.Req
NCIA: send CLS_CONNECT_CNF to DLU
NCIA: Receive Flow.Req
```

In the next two messages, the host sends data to the client:

```
NCIA: Receive Data.Req
NCIA: send CLS_DATA_IND to DLU
```

In the last two messages, the client closes the session:

```
NCIA: send CLS_DISC_IND to DLU
NCIA: Receive Disconnect.Rsp
```

Related Commands

```
debug dlsw
debug ncia circuit
debug ncia client
```

debug netbios error

Use the **debug netbios error** EXEC command to display information about Network Basic Input/Output System (NetBIOS) protocol errors. The **no** form of this command disables debugging output.

[no] debug netbios error

Usage Guidelines

For complete information on the NetBIOS process, use the **debug netbios packet** command along with the **debug netbios error** command.

Sample Display

The following is sample output from the **debug netbios error** command. This example shows that an illegal packet has been received on the async interface.

```
Router# debug netbios error  
  
Async1 nbf Bad packet
```

Related Commands

debug netbios-name-cache
debug netbios packet

debug netbios-name-cache

Use the **debug netbios-name-cache** EXEC command to display name caching activities on a router. The **no** form of this command disables debugging output.

[no] debug netbios-name-cache

Usage Guidelines

Examine the display to diagnose problems in NetBIOS name caching.

Sample Display

The following is sample output from the **debug netbios-name-cache** command:

```
Router# debug netbios-name-cache

NETBIOS: L checking name ORINDA, vrn=0
NETBIOS name cache table corrupted at offset 13
NETBIOS name cache table corrupted at later offset, at location 13
NETBIOS: U chk name=ORINDA, addr=1000.4444.5555, idb=TR1, vrn=0, type=1
NETBIOS: U upd name=ORINDA,addr=1000.4444.5555,idb=TR1,vrn=0,type=1
NETBIOS: U add name=ORINDA,addr=1000.4444.5555,idb=TR1,vrn=0,type=1
NETBIOS: U no memory to add cache entry. name=ORINDA,addr=1000.4444.5555
NETBIOS: Invalid structure detected in netbios_name_cache_ager
NETBIOS: flushed name=ORINDA, addr=1000.4444.5555
NETBIOS: expired name=ORINDA, addr=1000.4444.5555
NETBIOS: removing entry. name=ORINDA,addr=1000.4444.5555,idb=TR1,vrn=0
NETBIOS: Tossing ADD_NAME/STATUS/NAME/ADD_GROUP frame
NETBIOS: Lookup Failed -- not in cache
NETBIOS: Lookup Worked, but split horizon failed
NETBIOS: Could not find RIF entry
NETBIOS: Cannot duplicate packet in netbios_name_cache_proxy
```

Note The sample display is a composite output. Debugging output that you actually see would not necessarily occur in this sequence.

Table 95 describes selected output fields.

Table 95 Debug NetBIOS-Name-Cache Field Descriptions

| Field | Description |
|---------------------|--|
| NETBIOS | This is a NetBIOS name caching debugging output. |
| L, U | L means lookup; U means update. |
| addr=1000.4444.5555 | MAC address 1000.4444.5555 of machine being looked up in NetBIOS name cache. |
| idb=TR1 | Indication that name of machine was learned from Token Ring interface number 1; idb translates into interface data block. |
| vrn=0 | Router determined that the packet comes from virtual ring number 0; this packet actually comes from a real Token Ring interface, because virtual ring number 0 is not valid. |

Table 95 Debug NetBIOS-Name-Cache Field Descriptions (Continued)

| Field | Description |
|--------|--|
| type=1 | <p>The type field indicates the way that the router learned about the specified machine. The possible values for type are as follows:</p> <ul style="list-style-type: none"> • 1 = Learned from traffic • 2 = Learned from a remote peer • 4, 8 = Statically entered via the router's configuration |

With the first line of output, the router declares that it has examined the NetBIOS name cache table for the machine name ORINDA and that the packet that prompted the lookup came from virtual ring 0. In this case, this packet comes from a real interface—virtual ring number 0 is not valid.

```
NETBIOS: L checking name ORINDA, vrn=0
```

The following two lines indicate that an invalid NetBIOS entry exists and that the corrupted memory was detected. The invalid memory will be removed from the table; no action is needed.

```
NETBIOS name cache table corrupted at offset 13
NETBIOS name cache table corrupted at later offset, at location 13
```

The following line indicates that the router attempted to check the NetBIOS cache table for the name ORINDA with MAC address 1000.4444.5555. This name was obtained from Token Ring interface 1. The type field indicates that the name was learned from traffic.

```
NETBIOS: U chk name=ORINDA, addr=1000.4444.5555, idb=TR1, vrn=0, type=1
```

The following line indicates that the NetBIOS name ORINDA is in the name cache table and was updated to the current value:

```
NETBIOS: U upd name=ORINDA, addr=1000.4444.5555, idb=TR1, vrn=0, type=1
```

The following line indicates that the NetBIOS name ORINDA is not in the table and must be added to the table:

```
NETBIOS: U add name=ORINDA, addr=1000.4444.5555, idb=TR1, vrn=0, type=1
```

The following line indicates that there was insufficient cache buffer space when the router tried to add this name:

```
NETBIOS: U no memory to add cache entry. name=ORINDA, addr=1000.4444.5555
```

The following line indicates that the NetBIOS ager detects an invalid memory in the cache. The router clears the entry; no action is needed.

```
NETBIOS: Invalid structure detected in netbios_name_cache_ager
```

The following line indicates that the entry for ORINDA was flushed from the cache table:

```
NETBIOS: flushed name=ORINDA, addr=1000.4444.5555
```

The following line indicates that the entry for ORINDA timed out and was flushed from the cache table:

```
NETBIOS: expired name=ORINDA, addr=1000.4444.5555
```

The following line indicates that the router removed the ORINDA entry from its cache table:

```
NETBIOS: removing entry. name=ORINDA, addr=1000.4444.5555, idb=TR1, vrn=0
```

The following line indicates that the router discarded a NetBIOS packet of type ADD_NAME, STATUS, NAME_QUERY, or ADD_GROUP. These packets are discarded when multiple copies of one of these packet types are detected during a certain period of time.

```
NETBIOS: Tossing ADD_NAME/STATUS/NAME/ADD_GROUP frame
```

The following line indicates that the system could not find a NetBIOS name in the cache:

```
NETBIOS: Lookup Failed -- not in cache
```

The following line indicates that the system found the destination NetBIOS name in the cache, but located on the same ring from which the packet came. The router will drop this packet because the packet should not leave this ring.

```
NETBIOS: Lookup Worked, but split horizon failed
```

The following line indicates that the system found the NetBIOS name in the cache, but the router could not find the corresponding RIF. The packet will be sent as a broadcast frame.

```
NETBIOS: Could not find RIF entry
```

The following line indicates that no buffer was available to create a NetBIOS name-cache proxy. A proxy will not be created for the packet, which will be forwarded as a broadcast frame.

```
NETBIOS: Cannot duplicate packet in netbios_name_cache_proxy
```

Related Commands

debug netbios error
debug netbios packet

debug netbios packet

Use the **debug netbios packet** EXEC command to display general information about NetBIOS packets. The **no** form of this command disables debugging output.

[no] debug netbios packet

Usage Guidelines

For complete information on the NetBIOS process, use the **debug netbios error** command along with the **debug netbios packet** command.

Sample Display

The following is sample output from the **debug netbios packet** and **debug netbios error** commands. This example shows the LLC header for an asynchronous interface followed by the NetBIOS information. For additional information on the NetBIOS fields, refer to *IBM LAN Technical Reference IEEE 802.2*.

```
Router# debug netbios packet

Async1 (i) U-format UI C_R=0x0
(i) NETBIOS_ADD_NAME_QUERY
  Resp_correlator= 0x6F 0x0
  Src name=CS-NT-1

Async1 (i) U-format UI C_R=0x0
(i) NETBIOS_ADD_GROUP_QUERY
  Resp_correlator= 0x6F 0x0
  Src name=COMMSERVER-WG

Async1 (i) U-format UI C_R=0x0
(i) NETBIOS_ADD_NAME_QUERY
  Resp_correlator= 0x6F 0x0
  Src name=CS-NT-1

Ethernet0 (i) U-format UI C_R=0x0
(i) NETBIOS_DATAGRAM
  Length= 0x2C 0x0
  Dest name=COMMSERVER-WG
  Src name=CS-NT-3
```

Related Commands

debug netbios error
debug netbios-name-cache

debug nhrp

Use the **debug nhrp** EXEC command to display information about Next Hop Resolution Protocol (NHRP) activity. The **no** form of this command disables debugging output.

[no] debug nhrp

Usage Guidelines

Use this command when some nodes on a TCP/IP or IPX network are not responding. It shows whether the router is sending or receiving NHRP packets.

Sample Display

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

```
Router# debug nhrp

NHRP: Cache update 172.19.145.57 None
NHRP: Sent request src 172.19.145.56 dst 255.255.255.255
NHRP M: id 0 src 172.19.145.56 dst 172.19.145.57
NHRP: Encapsulation succeeded. MAC addr ffff.ffff.ffff.
NHRP: O 86 bytes out Ethernet1 dest 255.255.255.255
NHRP: Recv reply Size 64
NHRP M: id 0 src 172.19.145.56 dst 172.19.145.57
NHRP: Cache update 172.19.145.57 0000.0c14.59d3.
```

Table 96 describes the fields shown in the display.

Table 96 Debug NHRP Field Descriptions

| Field | Descriptions |
|--------------------------------------|---|
| NHRP and NHRP M | NHRP debugging output and mandatory header debugging output. |
| Cache update | NHRP cache is being revised. |
| Sent request src dst | NHRP request packet was sent from the specified source address. NHRP packet was sent to the specified destination address. |
| id | Sequence number of the packet. |
| src | Sequence number of the source address. |
| dst | Sequence number of the destination address. |
| Encapsulation succeeded. MAC addr | NHRP packet was successfully encapsulated. Link layer address used as the destination address for the NHRP packet. |
| O 86 bytes out Ethernet1 dest | Size of the NHRP packet (in this case, the output was 86 bytes). Interface that the packet was sent out on, and the network layer destination address. |
| Recv reply Size | Indicates receipt of an NHRP reply packet and the size of the packet excluding the link layer header. |

Related Commands

debug nhrp options
debug nhrp packet

debug nhrp extension

To display the extensions portion of an NHRP packet, use the **debug nhrp extension EXEC** command. The **no** form of this command disables debugging output.

[no] debug nhrp extension

Sample Display

The following is sample output from the **debug nhrp extension** command:

```
Router# debug nhrp extension
NHRP extension processing debugging is on
Router#
Forward Transit NHS Record Extension(4):
(C-1) code: no error(0)
    prefix: 0, mtu: 9180, hd_time: 7200
    addr_len: 20(NSAP), subaddr_len: 0(NSAP), proto_len: 4, pref: 0
    client NBMA: 47.009181000000002ba08e101.525354555354.01
    client protocol: 135.206.58.54
Reverse Transit NHS Record Extension(5):
Responder Address Extension(3):
(C) code: no error(0)
    prefix: 0, mtu: 9180, hd_time: 7200
    addr_len: 20(NSAP), subaddr_len: 0(NSAP), proto_len: 4, pref: 0
    client NBMA: 47.009181000000002ba08e101.525354555355.01
    client protocol: 135.206.58.55
Forward Transit NHS Record Extension(4):
(C-1) code: no error(0)
    prefix: 0, mtu: 9180, hd_time: 7200
    addr_len: 20(NSAP), subaddr_len: 0(NSAP), proto_len: 4, pref: 0
    client NBMA: 47.009181000000002ba08e101.525354555354.01
    client protocol: 135.206.58.54
Reverse Transit NHS Record Extension(5):
Responder Address Extension(3):
Forward Transit NHS Record Extension(4):
Reverse Transit NHS Record Extension(5):
```

debug nhrp options

Use the **debug nhrp options** EXEC command to display information about NHRP option processing. The **no** form of this command disables debugging output.

[no] debug nhrp options

Usage Guidelines

Use this command to show you whether there are problems or error situations with NHRP option processing (for example, unknown options).

Sample Display

The following is sample output from the **debug nhrp options** command:

```
Router# debug nhrp options

NHRP-OPT: MASK 4
NHRP-OPT-MASK: FFFFFFFF
NHRP-OPT: NETID 4
NHRP-OPT: RESPONDER 4
NHRP-OPT: RECORD 0
NHRP-OPT: RRECORD 0
```

Table 97 describes the fields.

Table 97 **Debug NHRP Options Field Descriptions**

| Field | Descriptions |
|---------------|---|
| NHRP-OPT | NHRP options debugging output. |
| MASK 4 | Number of bytes of information in the destination prefix option. |
| NHRP-OPT-MASK | Contents of the destination prefix option. |
| NETID | Number of bytes of information in the subnetwork identifier option. |
| RESPONDER | Number of bytes of information in the responder address option. |
| RECORD | Forward record option. |
| RRECORD | Reverse record option. |

Related Commands

debug nhrp
debug nhrp packet

debug nhrp packet

To display a dump of NHRP packets, use the **debug nhrp packet EXEC** command. The **no** form of this command disables debugging output.

[no] debug nhrp packet

Sample Display

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

```
Router# debug nhrp packet
NHRP activity debugging is on
Router#
NHRP: Send Purge Request via ATM3/0.1, packet size: 72
src: 135.206.58.55, dst: 135.206.58.56
(F) afn: NSAP(3), type: IP(800), hop: 255, ver: 1
    shtl: 20(NSAP), sstl: 0(NSAP)
(M) flags: "reply required", reqid: 2
    src NBMA: 47.009181000000002ba08e101.525354555355.01
    src protocol: 135.206.58.55, dst protocol: 135.206.58.56
(C-1) code: no error(0)
    prefix: 0, mtu: 9180, hd_time: 0
    addr_len: 0(NSAP), subaddr_len: 0(NSAP), proto_len: 4, pref: 0
    client protocol: 135.206.58.130
NHRP: Receive Purge Reply via ATM3/0.1, packet size: 72
(F) afn: NSAP(3), type: IP(800), hop: 254, ver: 1
    shtl: 20(NSAP), sstl: 0(NSAP)
(M) flags: "reply required", reqid: 2
    src NBMA: 47.009181000000002ba08e101.525354555355.01
    src protocol: 135.206.58.55, dst protocol: 135.206.58.56
(C-1) code: no error(0)
    prefix: 0, mtu: 9180, hd_time: 0
    addr_len: 0(NSAP), subaddr_len: 0(NSAP), proto_len: 4, pref: 0
    client protocol: 135.206.58.130
```

debug nhrp rate

Use the **debug nhrp rate** EXEC command to display information about NHRP traffic rate limits. The **no** form of this command disables debugging output.

[no] debug nhrp rate

Usage Guidelines

Use this command to verify that the traffic is consistent with the setting of the NHRP commands (such as **ip nhrp use** and **ip max-send** commands).

Sample Display

The following is sample output from the **debug nhrp rate** command:

```
Router# debug nhrp rate

NHRP-RATE: Sending initial request
NHRP-RATE: Retransmitting request (retrans ivl 2)
NHRP-RATE: Retransmitting request (retrans ivl 4)
NHRP-RATE: Ethernet1: Used 3
```

Table 98 describes the fields.

Table 98 Debug NHRP Rate Field Descriptions

| Field | Descriptions |
|-------------------------|---|
| NHRP-RATE | NHRP rate debugging output. |
| Sending initial request | First time an attempt was made to send an NHRP packet to a particular destination. |
| Retransmitting request | Indicates that the NHRP packet was retransmitted, and shows the time interval (in seconds) to wait before the NHRP packet is retransmitted again. |
| Ethernet1: | Interface over which the NHRP packet was transmitted. |
| Used 3 | Number of packets sent out of the default maximum 5 (in this case, 3 were sent). |

Related Commands

debug nhrp
debug nhrp options

debug packet

Use the **debug packet** EXEC command to display information on packets that the network can not classify. The **no** form of this command disables debugging output.

[no] debug packet

Sample Display

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

```
Router# debug packet

Ethernet0: Unknown ARPA, src 0000.0c00.6fa4, dst ffff.ffff.ffff, type 0x0a0
data 00000c00f23a00000c00ab45, len 60
Serial3: Unknown HDLC, size 64, type 0xaaaa, flags 0x0F00
Serial2: Unknown PPP, size 128
Serial7: Unknown FRAME-RELAY, size 174, type 0x5865, DLCI 7a
Serial0: compressed TCP/IP packet dropped
```

Table 99 describes significant fields.

Table 99 **Debug Packet Field Descriptions**

| Field | Description |
|-----------|---|
| Ethernet0 | Name of the Ethernet interface that received the packet. |
| Unknown | The network could not classify this packet. Examples include packets with unknown link types. |
| ARPA | <p>This packet uses ARPA-style encapsulation. Possible encapsulation styles vary depending on the media command mode (MCM) and encapsulation style, as follows:</p> <p>Ethernet (MCM)—Encapsulation Style</p> <ul style="list-style-type: none"> • APOLLO • ARP • ETHERTALK • ISO1 • ISO3 • LLC2 • NOVELL-ETHER • SNAP <hr/> <p>FDDI (MCM)—Encapsulation Style</p> <ul style="list-style-type: none"> • APOLLO • ISO1 • ISO3 • LLC2 • SNAP <hr/> <p>Frame Relay—Encapsulation Style</p> <ul style="list-style-type: none"> • BRIDGE • FRAME-RELAY |

Table 99 **Debug Packet Field Descriptions (Continued)**

| Field | Description |
|----------------------------------|--|
| | Serial (MCM)—Encapsulation Style <ul style="list-style-type: none"> • BFEX25 • BRIDGE • DDN-X25 • DDNX25-DCE • ETHERTALK • FRAME-RELAY • HDLC • HDH • LAPB • LAPBDCE • MULTI-LAPB • PPP • SDLC-PRIMARY • SDLC-SECONDARY • SLIP • SMDS • STUN • X25 • X25-DCE |
| | Token Ring (MCM)—Encapsulation Style <ul style="list-style-type: none"> • 3COM-TR • ISO1 • ISO3 • MAC • LLC2 • NOVELL-TR • SNAP • VINES-TR |
| src 0000.0c00.6fa4 | MAC address of the node generating the packet. |
| dst.ffff.ffff.ffff | MAC address of the destination node for the packet. |
| type 0x0a0 | Packet type. |
| data... | First 12 bytes of the datagram following the MAC header. |
| len 60 | Length of the message in bytes that the interface received from the wire. |
| size 64 | Length of the message in bytes that the interface received from the wire. Equivalent to the len field. |
| flags 0x0F00 | HDLC or PP flags field. |
| DLCI 7a | The DLCI number on Frame Relay. |
| compressed TCP/IP packet dropped | This message can occur when TCP header compression is enabled on an interface and the packet does not turn out to be HDLC or X25 after classification. |

debug ppp

Use the **debug ppp** EXEC command to display information on traffic and exchanges in an internetwork implementing the Point-to-Point Protocol (PPP). The **no** form of this command disables debugging output.

[no] debug ppp { packet | negotiation | error | authentication | compression | cbcp }

Syntax Description

| | |
|-----------------------|---|
| packet | Causes the debug ppp command to display PPP packets being sent and received. (This command displays low-level packet dumps.) |
| negotiation | Causes the debug ppp command to display PPP packets transmitted during PPP startup, where PPP options are negotiated. |
| error | Causes the debug ppp command to display protocol errors and error statistics associated with PPP connection negotiation and operation. |
| authentication | Causes the debug ppp command to display authentication protocol messages, including Challenge Authentication Protocol (CHAP) packet exchanges and Password Authentication Protocol (PAP) exchanges. |
| compression | Causes the debug ppp command to display information specific to the exchange of PPP connections using MPPC. This command is useful for obtaining incorrect packet sequence number information where MPPC compression is enabled. |
| cbcp | Causes the debug ppp command to display protocol errors and statistics associated with PPP connection negotiations using MSCB. |

Usage Guidelines

Use the **debug ppp** commands when trying to find the following:

- The Network Control Protocols (NCPs) that are supported on either end of a PPP connection
- Any loops that might exist in a PPP internetwork
- Nodes that are (or are not) properly negotiating PPP connections
- Errors that have occurred over the PPP connection
- Causes for CHAP session failures
- Causes for PAP session failures
- Information specific to the exchange of PPP connections using the Callback Control Protocol (CBCP), used by Microsoft clients.
- Incorrect packet sequence number information where MPPC compression is enabled.

Refer to Internet RFCs 1331, 1332, and 1333 for details concerning PPP-related nomenclature and protocol information.



Caution The **debug ppp compression** command is CPU-intensive and should be used with caution. This command should be disabled immediately after debugging.

Sample Displays

The following is sample output from the **debug ppp packet** command as seen from the Link Quality Monitor (LQM) side of the connection. This display example depicts packet exchanges under normal PPP operation.

```
Router# debug ppp packet

PPP Serial4(o): lcp_slqr() state = OPEN magic = D21B4, len = 48
PPP Serial4(i): pkt type 0xC025, datagramsize 52
PPP Serial4(i): lcp_rlqr() state = OPEN magic = D3454, len = 48
PPP Serial4(i): pkt type 0xC021, datagramsize 16
PPP Serial4: I LCP ECHOREQ(9) id 3 (C) magic D3454
PPP Serial4: input(C021) state = OPEN code = ECHOREQ(9) id = 3 len = 12
PPP Serial4: O LCP ECHOREP(A) id 3 (C) magic D21B4
PPP Serial4(o): lcp_slqr() state = OPEN magic = D21B4, len = 48
PPP Serial4(i): pkt type 0xC025, datagramsize 52
PPP Serial4(i): lcp_rlqr() state = OPEN magic = D3454, len = 48
PPP Serial4(i): pkt type 0xC021, datagramsize 16
PPP Serial4: I LCP ECHOREQ(9) id 4 (C) magic D3454
PPP Serial4: input(C021) state = OPEN code = ECHOREQ(9) id = 4 len = 12
PPP Serial4: O LCP ECHOREP(A) id 4 (C) magic D21B4
PPP Serial4(o): lcp_slqr() state = OPEN magic = D21B4, len = 48
PPP Serial4(i): pkt type 0xC025, datagramsize 52
PPP Serial4(i): lcp_rlqr() state = OPEN magic = D3454, len = 48
PPP Serial4(i): pkt type 0xC021, datagramsize 16
PPP Serial4: I LCP ECHOREQ(9) id 5 (C) magic D3454
PPP Serial4: input(C021) state = OPEN code = ECHOREQ(9) id = 5 len = 12
PPP Serial4: O LCP ECHOREP(A) id 5 (C) magic D21B4
PPP Serial4(o): lcp_slqr() state = OPEN magic = D21B4, len = 48
PPP Serial4(i): pkt type 0xC025, datagramsize 52
PPP Serial4(i): lcp_rlqr() state = OPEN magic = D3454, len = 48
PPP Serial4(i): pkt type 0xC021, datagramsize 16
PPP Serial4: I LCP ECHOREQ(9) id 6 (C) magic D3454
PPP Serial4: input(C021) state = OPEN code = ECHOREQ(9) id = 6 len = 12
PPP Serial4: O LCP ECHOREP(A) id 6 (C) magic D21B4
PPP Serial4(o): lcp_slqr() state = OPEN magic = D21B4, len = 48
PPP Serial4(i): pkt type 0xC025, datagramsize 52
PPP Serial4(i): lcp_rlqr() state = OPEN magic = D3454, len = 48
PPP Serial4(i): pkt type 0xC021, datagramsize 16
PPP Serial4: I LCP ECHOREQ(9) id 7 (C) magic D3454
PPP Serial4: input(C021) state = OPEN code = ECHOREQ(9) id = 7 len = 12
PPP Serial4: O LCP ECHOREP(A) id 7 (C) magic D21B4
PPP Serial4(o): lcp_slqr() state = OPEN magic = D21B4, len = 48
```

Table 100 describes significant fields in the output.

Table 100 **Debug PPP Packet Field Descriptions**

| Field | Description |
|--------------|---|
| PPP | This is PPP debugging output. |
| Serial4 | Interface number associated with this debugging information. |
| (o), O | This packet was detected as an output packet. |
| (i), I | This packet was detected as an input packet. |
| lcp_slqr() | Procedure name; running LQM, send a Link Quality Report (LQR). |
| lcp_rlqr() | Procedure name; running LQM, received an LQR. |
| input (C021) | The router received a packet of the specified packet type (in hexadecimal). A value of C025 indicates packet of type LQM. |

Table 100 Debug PPP Packet Field Descriptions (Continued)

| Field | Description |
|-------------------|---|
| state = OPEN | PPP state; normal state is OPEN. |
| magic = D21B4 | Magic Number for indicated node; when output is indicated, this is the Magic Number of the node on which debugging is enabled. The actual Magic Number depends on whether the packet detected is indicated as I or O. |
| datagramsize = 52 | Packet length including header. |
| code = ECHOREQ(9) | Code identifies the type of packet received. Both forms of the packet, string and hexadecimal, are presented. |
| len = 48 | Packet length without header. |
| id = 3 | ID number per Link Control Protocol (LCP) packet format. |
| pkt type 0xC025 | Packet type in hexadecimal; typical packet types are C025 for LQM and C021 for LCP. |
| LCP ECHOREQ (9) | Echo Request; value in parentheses is the hexadecimal representation of the LCP type. |
| LCP ECHOREP (A) | Echo Reply; value in parentheses is the hexadecimal representation of the LCP type. |

To elaborate on the displayed output, consider the partial exchange. This sequence shows that one side is using ECHO for its keepalives and the other side is using LQRs.

```
Router# debug ppp packet
```

```
PPP Serial4(o): lcp_slqr() state = OPEN magic = D21B4, len = 48
PPP Serial4(i): pkt type 0xC025, datagramsize 52
PPP Serial4(i): lcp_rlqr() state = OPEN magic = D3454, len = 48
PPP Serial4(i): pkt type 0xC021, datagramsize 16
PPP Serial4: I LCP ECHOREQ(9) id 3 (C) magic D3454
PPP Serial4: input(C021) state = OPEN code = ECHOREQ(9) id = 3 len = 12
PPP Serial4: O LCP ECHOREP(A) id 3 (C) magic D21B4
PPP Serial4(o): lcp_slqr() state = OPEN magic = D21B4, len = 48
```

The first line states that the router with debugging enabled has sent an LQR to the other side of the PPP connection:

```
PPP Serial4(o): lcp_slqr() state = OPEN magic = D21B4, len = 48
```

The next two lines indicate that the router has received a packet of type C025 (LQM) and provides details about the packet:

```
PPP Serial4(i): pkt type 0xC025, datagramsize 52
PPP Serial4(i): lcp_rlqr() state = OPEN magic = D3454, len = 48
```

The next two lines indicate that the router received an ECHOREQ of type C021 (LCP). The other side is sending ECHOs. The router on which debugging is configured for LQM but also responds to ECHOs.

```
PPP Serial4(i): pkt type 0xC021, datagramsize 16
PPP Serial4: I LCP ECHOREQ(9) id 3 (C) magic D3454
```

Next, the router is detected to have responded to the ECHOREQ with an ECHOREP and is preparing to send out an LQR:

```
PPP Serial4: O LCP ECHOREP(A) id 3 (C) magic D21B4
PPP Serial4(o): lcp_slqr() state = OPEN magic = D21B4, len = 48
```

The following is sample output from the **debug ppp negotiation** command. This is a normal negotiation, where both sides agree on network control program (NCP) parameters. In this case, protocol type IP is proposed and acknowledged.

```
Router# debug ppp negotiation

ppp: sending CONFREQ, type = 4 (CI_QUALITYTYPE), value = C025/3E8
ppp: sending CONFREQ, type = 5 (CI_MAGICNUMBER), value = 3D56CAC
ppp: received config for type = 4 (QUALITYTYPE) acked
ppp: received config for type = 5 (MAGICNUMBER) value = 3D567F8 acked (ok)
PPP Serial4: state = ACKSENT fsm_rconfack(C021): rcvd id 5
ppp: config ACK received, type = 4 (CI_QUALITYTYPE), value = C025
ppp: config ACK received, type = 5 (CI_MAGICNUMBER), value = 3D56CAC
ppp: ipcp_reqci: returning CONFACK.
      (ok)
PPP Serial4: state = ACKSENT fsm_rconfack(8021): rcvd id 4
```

Table 101 describes significant fields in the output.

Table 101 Debug PPP Negotiation Field Descriptions

| Field | Description |
|---------------------------|--|
| ppp | This is PPP debugging output. |
| sending CONFREQ | The router sent a configuration request. |
| type = 4 (CI_QUALITYTYPE) | The type of LCP configuration option that is being negotiated and a descriptor. A type value of 4 indicates Quality Protocol negotiation; a type value of 5 indicates Magic Number negotiation. |
| value = C025/3E8 | For Quality Protocol negotiation, indicates NCP type and reporting period. In the example, C025 indicates LQM; 3E8 is a hexadecimal value translating to about 10 seconds (in hundredths of a second). |
| value = 3D56CAC | For Magic Number negotiation, indicates the Magic Number being negotiated. |
| received config | The receiving node has received the proposed option negotiation for the indicated option type. |
| acked | Acknowledgment and acceptance of options. |
| state = ACKSENT | Specific PPP state in the negotiation process. |
| ipcp_reqci | IPCP notification message; sending CONFACK. |
| fsm_rconfack (8021) | The procedure fsm_rconfack processes received CONFACKs, and the protocol (8021) is IP. |

The first two lines indicate that the router is trying to bring up LCP and will use the indicated negotiation options (Quality Protocol and Magic Number). The value fields are the values of the options themselves. C025/3E8 translates to Quality Protocol LQM. 3E8 is the reporting period (in hundredths of a second). 3D56CAC is the value of the Magic Number for the router.

```
ppp: sending CONFREQ, type = 4 (CI_QUALITYTYPE), value = C025/3E8
ppp: sending CONFREQ, type = 5 (CI_MAGICNUMBER), value = 3D56CAC
```

The next two lines indicate that the other side negotiated for options 4 and 5 as requested and acknowledged both. If the responding end does not support the options, a CONFREJ is sent by the responding node. If the responding end does not accept the value of the option, a CONFNAK is sent with the value field modified.

```
ppp: received config for type = 4 (QUALITYTYPE) acked
ppp: received config for type = 5 (MAGICNUMBER) value = 3D567F8 acked (ok)
```

The next three lines indicate that the router received a CONFACK from the responding side and displays accepted option values. Use the rcvd id field to verify that the CONFREQ and CONFACK have the same id field.

```
PPP Serial4: state = ACKSENT fsm_rconfack(C021): rcvd id 5
ppp: config ACK received, type = 4 (CI_QUALITYTYPE), value = C025
ppp: config ACK received, type = 5 (CI_MAGICNUMBER), value = 3D56CAC
```

The next line indicates that the router has IP routing enabled on this interface and that the IPCP NCP negotiated successfully:

```
ppp: ipcp_reqci: returning CONFACK.
```

In the last line, the router's state is listed as ACKSENT.

```
PPP Serial4: state = ACKSENT fsm_rconfack(C021): rcvd id 5\
```

The following is sample output from when the **debug ppp packet** and **debug ppp negotiation** commands are enabled at the same time.

```
router# debug ppp negotiation
router# debug ppp packet
```

```
ppp: sending CONFREQ, type = 4 (CI_QUALITYTYPE), value = C025/3E8
ppp: sending CONFREQ, type = 5 (CI_MAGICNUMBER), value = D4C64
PPP Serial4: O LCP CONFREQ(1) id 4 (12) QUALITYTYPE (8) 192 37 0 0 3 232
MAGICNUMBER (6) 0 13 76 100
PPP Serial4(i): pkt type 0xC021, datagramsize 22
PPP Serial4: I LCP CONFREQ(1) id 4 (12) QUALITYTYPE (8) 192 37 0 0 3 232
MAGICNUMBER (6) 0 13 84 240
PPP Serial4: input(C021) state = REQSENT code = CONFREQ(1) id = 4 len = 18
ppp: received config for type = 4 (QUALITYTYPE) acked
ppp: received config for type = 5 (MAGICNUMBER) value = D54F0 acked
PPP Serial4: O LCP CONFACK(2) id 4 (12) QUALITYTYPE (8) 192 37 0 0 3 232
MAGICNUMBER (6) 0 13 84 240 (ok)
PPP Serial4(i): input(C021) state = REQSENT code = CONFREQ(1) id = 4 len = 18
PPP Serial4: I LCP CONFACK(2) id 4 (12) QUALITYTYPE (8) 192 37 0 0 3 232
MAGICNUMBER (6) 0 13 76 100
PPP Serial4: input(C021) state = ACKSENT code = CONFACK(2) id = 4 len = 18
PPP Serial4: state = ACKSENT fsm_rconfack(C021): rcvd id 4
ppp: config ACK received, type = 4 (CI_QUALITYTYPE), value = C025
ppp: config ACK received, type = 5 (CI_MAGICNUMBER), value = D4C64
ipcp: sending CONFREQ, type = 3 (CI_ADDRESS), Address = 2.1.1.2
PPP Serial4: O IPCP CONFREQ(1) id 3 (10) Type3 (6) 2 1 1 2
PPP Serial4: I IPCP CONFREQ(1) id 3 (10) Type3 (6) 2 1 1 1
PPP Serial4(i): pkt type 0x8021, datagramsize 14
PPP Serial4: input(8021) state = REQSENT code = CONFREQ(1) id = 3 len = 10
ppp Serial4: Negotiate IP address: her address 2.1.1.1 (ACK)
ppp: ipcp_reqci: returning CONFACK.
PPP Serial4: O IPCP CONFACK(2) id 3 (10) Type3 (6) 2 1 1 1 (ok)
PPP Serial4: I IPCP CONFACK(2) id 3 (10) Type3 (6) 2 1 1 2
PPP Serial4: input(8021) state = ACKSENT code = CONFACK(2) id = 3 len = 10
PPP Serial4: state = ACKSENT fsm_rconfack(8021): rcvd id 3
ipcp: config ACK received, type = 3 (CI_ADDRESS), Address = 2.1.1.2
PPP Serial4(o): lcp_slqr() state = OPEN magic = D4C64, len = 48
PPP Serial4(i): pkt type 0xC025, datagramsize 52
PPP Serial4(i): lcp_rlqr() state = OPEN magic = D54F0, len = 48
PPP Serial4(i): pkt type 0xC025, datagramsize 52
PPP Serial4(i): lcp_rlqr() state = OPEN magic = D54F0, len = 48
PPP Serial4(o): lcp_slqr() state = OPEN magic = D4C64, len = 48
```

This field shows a decimal representation of the Magic Number.

This field shows a decimal representation of the NCP value.

This field shows a decimal representation of the reporting period.

This exchange represents a successful PPP negotiation for support of NCP type IPCP.

S2877

The following is sample output from the **debug ppp error** command. These messages might appear when the Quality Protocol option is enabled on an interface that is already running PPP.

```
Router# debug ppp error

PPP Serial3(i): rlqr receive failure. successes = 15
PPP: myrcvdiffp = 159 peerxmitdiffp = 41091
PPP: myrcvdiffo = 2183 peerxmitdiffo = 1714439
PPP: threshold = 25
PPP Serial4(i): rlqr transmit failure. successes = 15
PPP: myxmitdiffp = 41091 peerrcvdiffp = 159
PPP: myxmitdiffo = 1714439 peerrcvdiffo = 2183
PPP: 1->OutLQRs = 1 LastOutLQRs = 1
PPP: threshold = 25
PPP Serial3(i): lqr_protrej() Stop sending LQRs.
PPP Serial3(i): The link appears to be looped back.
```

Table 102 describes significant fields in the output.

Table 102 Debug PPP Error Field Descriptions

| Field | Description |
|-------------------------|--|
| PPP | This is PPP debugging output. |
| Serial3(i) | Interface number associated with this debugging information; indicates that this is an input packet. |
| rlqr receive failure | The request to negotiate the Quality Protocol option is not accepted. |
| myrcvdiffp = 159 | Number of packets received over the time period. |
| peerxmitdiffp = 41091 | Number of packets sent by the remote node over this period. |
| myrcvdiffo = 2183 | Number of octets received over this period. |
| peerxmitdiffo = 1714439 | Number of octets sent by the remote node over this period. |
| threshold = 25 | The maximum error percentage acceptable on this interface. This percentage is calculated by the threshold value entered in the ppp quality number interface configuration command. A value of 100-number (100 minus <i>number</i>) is the maximum error percentage. In this case, a <i>number</i> of 75 was entered. This means that the local router must maintain a minimum 75 percent non-error percentage, or the PPP link will be considered down. |
| OutLQRs = 1 | Local router's current send LQR sequence number. |
| LastOutLQRs = 1 | The last sequence number that the remote node side has seen from the local node. |

The following is sample output from the **debug ppp authentication** command. Use this **debug** command to determine why an authentication fails.

```
Router# debug ppp authentication

Serial0: Unable to authenticate. No name received from peer
Serial0: Unable to validate CHAP response. USERNAME pioneer not found.
Serial0: Unable to validate CHAP response. No password defined for USERNAME pioneer
Serial0: Failed CHAP authentication with remote.
Remote message is Unknown name
Serial0: remote passed CHAP authentication.
Serial0: Passed CHAP authentication with remote.
Serial0: CHAP input code = 4 id = 3 len = 48
```

In general, these messages are self-explanatory. Fields that can show optional output are outlined in Table 103.

Table 103 Debug PPP Authentication Field Descriptions

| Field | Description |
|--------------------------------|---|
| Serial0 | Interface number associated with this debugging information and CHAP access session in question. |
| USERNAME pioneer not found. | The name <i>pioneer</i> in this example is the name received in the CHAP response. The router looks up this name in the list of usernames that are configured for the router. |
| Remote message is Unknown name | The following messages can appear: <ul style="list-style-type: none"> • No name received to authenticate • Unknown name • No secret for given name • Short MD5 response received • MD compare failed |
| code = 4 | Specific CHAP type packet detected. Possible values are as follows: <ul style="list-style-type: none"> • 1 = Challenge • 2 = Response • 3 = Success • 4 = Failure |
| id = 3 | ID number per Link Control Protocol (LCP) packet format. |
| len = 48 | Packet length without header. |

The following shows sample output from the **debug ppp** command using the **cbcp** keyword. This output depicts packet exchanges under normal PPP operation where the Cisco access server is waiting for the remote PC to respond to the MSCB request. The router also has **debug ppp negotiation** and **service timestamps msec** commands enabled.

```

Router# debug ppp cbcp

Dec 17 00:48:11.302: As8 MCB: User mscb Callback Number - Client ANY
Dec 17 00:48:11.306: Async8 PPP: 0 MCB Request(1) id 1 len 9
Dec 17 00:48:11.310: Async8 MCB: 0 1 1 0 9 2 5 0 1 0
Dec 17 00:48:11.314: As8 MCB: 0 Request Id 1 Callback Type Client-Num delay 0
Dec 17 00:48:13.342: As8 MCB: Timeout in state WAIT_RESPONSE
Dec 17 00:48:13.346: Async8 PPP: 0 MCB Request(1) id 2 len 9
Dec 17 00:48:13.346: Async8 MCB: 0 1 2 0 9 2 5 0 1 0
Dec 17 00:48:13.350: As8 MCB: 0 Request Id 2 Callback Type Client-Num delay 0
Dec 17 00:48:15.370: As8 MCB: Timeout in state WAIT_RESPONSE
Dec 17 00:48:15.374: Async8 PPP: 0 MCB Request(1) id 3 len 9
Dec 17 00:48:15.374: Async8 MCB: 0 1 3 0 9 2 5 0 1 0
Dec 17 00:48:15.378: As8 MCB: 0 Request Id 3 Callback Type Client-Num delay 0
Dec 17 00:48:17.398: As8 MCB: Timeout in state WAIT_RESPONSE
Dec 17 00:48:17.402: Async8 PPP: 0 MCB Request(1) id 4 len 9
Dec 17 00:48:17.406: Async8 MCB: 0 1 4 0 9 2 5 0 1 0
Dec 17 00:48:17.406: As8 MCB: 0 Request Id 4 Callback Type Client-Num delay 0
Dec 17 00:48:19.426: As8 MCB: Timeout in state WAIT_RESPONSE
Dec 17 00:48:19.430: Async8 PPP: 0 MCB Request(1) id 5 len 9
Dec 17 00:48:19.430: Async8 MCB: 0 1 5 0 9 2 5 0 1 0
Dec 17 00:48:19.434: As8 MCB: 0 Request Id 5 Callback Type Client-Num delay 0
Dec 17 00:48:21.454: As8 MCB: Timeout in state WAIT_RESPONSE
  
```

```

Dec 17 00:48:21.458: Async8 PPP: O MCB Request(1) id 6 len 9
Dec 17 00:48:21.462: Async8 MCB: O 1 6 0 9 2 5 0 1 0
Dec 17 00:48:21.462: As8 MCB: O Request Id 6 Callback Type Client-Num delay 0
Dec 17 00:48:23.482: As8 MCB: Timeout in state WAIT_RESPONSE
Dec 17 00:48:23.486: Async8 PPP: O MCB Request(1) id 7 len 9
Dec 17 00:48:23.490: Async8 MCB: O 1 7 0 9 2 5 0 1 0
Dec 17 00:48:23.490: As8 MCB: O Request Id 7 Callback Type Client-Num delay 0
Dec 17 00:48:25.510: As8 MCB: Timeout in state WAIT_RESPONSE
Dec 17 00:48:25.514: Async8 PPP: O MCB Request(1) id 8 len 9
Dec 17 00:48:25.514: Async8 MCB: O 1 8 0 9 2 5 0 1 0
Dec 17 00:48:25.518: As8 MCB: O Request Id 8 Callback Type Client-Num delay 0
Dec 17 00:48:26.242: As8 PPP: I pkt type 0xC029, datagramsize 18
Dec 17 00:48:26.246: Async8 PPP: I MCB Response(2) id 8 len 16
Dec 17 00:48:26.250: Async8 MCB: I 2 8 0 10 2 C C 1 32 34 39 32 36 31 33
0
Dec 17 00:48:26.254: As8 MCB: Received response
Dec 17 00:48:26.258: As8 MCB: Response CBK-Client-Num 2 12 12,  addr 1-2492613
Dec 17 00:48:26.262: Async8 PPP: O MCB Ack(3) id 9 len 16
Dec 17 00:48:26.266: Async8 MCB: O 3 9 0 10 2 C C 1 32 34 39 32 36 31 33
0
Dec 17 00:48:26.270: As8 MCB: O Ack Id 9 Callback Type Client-Num delay 12
Dec 17 00:48:26.270: As8 MCB: Negotiated MCB with peer
Dec 17 00:48:26.390: As8 LCP: I TERMREQ [Open] id 4 len 8 (0x00000000)
Dec 17 00:48:26.390: As8 LCP: O TERMACK [Open] id 4 len 4
Dec 17 00:48:26.394: As8 MCB: Peer terminating the link
Dec 17 00:48:26.402: As8 MCB: Initiate Callback for mscb at 2492613 using Async

```

The following is sample output from the **debug ppp compression** command with **service timestamps** enabled and shows a typical PPP packet exchange between the router and Microsoft client where the MPPC header sequence numbers increment correctly.

```

Router# debug ppp compression

00:04:14: BR0:1 MPPC: Decomp - hdr/exp_cc# 0x2003/0x0003
00:04:14: BR0:1 MPPC: Decomp - hdr/exp_cc# 0x2004/0x0004
00:04:14: BR0:1 MPPC: Decomp - hdr/exp_cc# 0x2005/0x0005
00:04:14: BR0:1 MPPC: Decomp - hdr/exp_cc# 0x2006/0x0006
00:04:14: BR0:1 MPPC: Decomp - hdr/exp_cc# 0x2007/0x0007

```

Table 104 describes the fields for the **debug ppp compression** output.

Table 104 Debug PPP Compression Fields

| Field | Description |
|---------------|--|
| Interface | Interface enabled with MPPC. |
| Decomp - hdr/ | Decompression header and bit settings. |
| exp_cc# | Expected coherency count. |
| 0x2003 | Received sequence number. |
| 0x0003 | Expected sequence number. |

The following shows sample output from **debug ppp negotiation** and **debug ppp error** commands, which can be used to troubleshoot initial ppp negotiation and setup errors. This example shows a virtual interface (virtual interface 1) during normal PPP operation and CCP negotiation.

```

Router# debug ppp nego error
Vt1 PPP: Unsupported or un-negotiated protocol. Link arp
VPDN: Chap authentication succeeded for p5200
Vi1 PPP: Phase is DOWN, Setup
Vi1 VPDN: Virtual interface created for dinesh@cisco.com

```

```
Vi1 VPDN: Set to Async interface
Vi1 PPP: Phase is DOWN, Setup
Vi1 VPDN: Clone from Vtemplate 1 filterPPP=0 blocking
Vi1 CCP: Re-Syncing history using legacy method
%LINK-3-UPDOWN: Interface Virtual-Access1, changed state to up
Vi1 PPP: Treating connection as a dedicated line
Vi1 PPP: Phase is ESTABLISHING, Active Open
Vi1 LCP: O CONFREQ [Closed] id 1 len 25
Vi1 LCP:   ACCM 0x000A0000 (0x0206000A0000)
Vi1 LCP:   AuthProto CHAP (0x0305C22305)
Vi1 LCP:   MagicNumber 0x000FB69F (0x0506000FB69F)
Vi1 LCP:   PFC (0x0702)
Vi1 LCP:   ACFC (0x0802)
Vi1 VPDN: Bind interface direction=2
Vi1 PPP: Treating connection as a dedicated line
Vi1 LCP: I FORCED CONFREQ len 21
Vi1 LCP:   ACCM 0x000A0000 (0x0206000A0000)
Vi1 LCP:   AuthProto CHAP (0x0305C22305)
Vi1 LCP:   MagicNumber 0x12A5E4B5 (0x050612A5E4B5)
Vi1 LCP:   PFC (0x0702)
Vi1 LCP:   ACFC (0x0802)
Vi1 VPDN: PPP LCP accepted sent & rcv CONFACK
Vi1 PPP: Phase is AUTHENTICATING, by this end
Vi1 CHAP: O CHALLENGE id 1 len 27 from "l_4000"
Vi1 CHAP: I RESPONSE id 20 len 37 from "dinesh@cisco.com"
Vi1 CHAP: O SUCCESS id 20 len 4
Vi1 PPP: Phase is UP
Vi1 IPCP: O CONFREQ [Closed] id 1 len 10
Vi1 IPCP:   Address 15.2.2.3 (0x03060F020203)
Vi1 CCP: O CONFREQ [Not negotiated] id 1 len 10
Vi1 CCP:   MS-PPC supported bits 0x00000001 (0x120600000001)
Vi1 IPCP: I CONFREQ [REQsent] id 1 len 34
Vi1 IPCP:   Address 0.0.0.0 (0x030600000000)
Vi1 IPCP:   PrimaryDNS 0.0.0.0 (0x810600000000)
Vi1 IPCP:   PrimaryWINS 0.0.0.0 (0x820600000000)
Vi1 IPCP:   SecondaryDNS 0.0.0.0 (0x830600000000)
Vi1 IPCP:   SecondaryWINS 0.0.0.0 (0x840600000000)
Vi1 IPCP: Using the default pool
Vi1 IPCP: Pool returned 11.2.2.5
Vi1 IPCP: O CONFREQ [REQsent] id 1 len 16
Vi1 IPCP:   PrimaryWINS 0.0.0.0 (0x820600000000)
Vi1 IPCP:   SecondaryWINS 0.0.0.0 (0x840600000000)
Vi1 CCP: I CONFREQ [REQsent] id 1 len 15
Vi1 CCP:   MS-PPC supported bits 0x00000001 (0x120600000001)
Vi1 CCP:   Stacker history 1 check mode EXTENDED (0x1105000104)
Vi1 CCP: Already accepted another CCP option, rejecting this STACKER
Vi1 CCP: O CONFREQ [REQsent] id 1 len 9
Vi1 CCP:   Stacker history 1 check mode EXTENDED (0x1105000104)
Vi1 IPCP: I CONFACK [REQsent] id 1 len 10
Vi1 IPCP:   Address 15.2.2.3 (0x03060F020203)
Vi1 CCP: I CONFACK [REQsent] id 1 len 10
Vi1 CCP:   MS-PPC supported bits 0x00000001 (0x120600000001)
Vi1 CCP: I CONFREQ [ACKrcvd] id 2 len 10
Vi1 CCP:   MS-PPC supported bits 0x00000001 (0x120600000001)
Vi1 CCP: O CONFACK [ACKrcvd] id 2 len 10
Vi1 CCP:   MS-PPC supported bits 0x00000001 (0x120600000001)
Vi1 CCP: State is Open
Vi1 IPCP: I CONFREQ [ACKrcvd] id 2 len 22
Vi1 IPCP:   Address 0.0.0.0 (0x030600000000)
Vi1 IPCP:   PrimaryDNS 0.0.0.0 (0x810600000000)
Vi1 IPCP:   SecondaryDNS 0.0.0.0 (0x830600000000)
Vi1 IPCP: O CONFNAK [ACKrcvd] id 2 len 22
Vi1 IPCP:   Address 11.2.2.5 (0x03060B020205)
Vi1 IPCP:   PrimaryDNS 171.69.1.148 (0x8106AB450194)
Vi1 IPCP:   SecondaryDNS 171.69.2.132 (0x8306AB450284)
```

```
Vi1 IPCP: I CONFREQ [ACKrcvd] id 3 len 22
Vi1 IPCP:   Address 11.2.2.5 (0x03060B020205)
Vi1 IPCP:   PrimaryDNS 171.69.1.148 (0x8106AB450194)
Vi1 IPCP:   SecondaryDNS 171.69.2.132 (0x8306AB450284)
Vi1 IPCP: O CONFACK [ACKrcvd] id 3 len 22
Vi1 IPCP:   Address 11.2.2.5 (0x03060B020205)
Vi1 IPCP:   PrimaryDNS 171.69.1.148 (0x8106AB450194)
Vi1 IPCP:   SecondaryDNS 171.69.2.132 (0x8306AB450284)
Vi1 IPCP: State is Open
Vi1 IPCP: Install route to 11.2.2.5
```