Ethernet Diagnostics

IRM options

Robert Goodwin

Thu, Oct 3, 2002

The IRM ethernet driver supports several diagnostics that can assist diagnosing network problems. This note summarizes various diagnostics that are available in these 68040-based front ends. Most of these tools are unavailable in the PowerPC nodes, because the vxWorks kernel supports all network activity. The exceptions are noted in the text.

Network Frames

All nodes are set up to include the `NETFRAME` data stream support, which records all network activity seen by the node. One usually views this data via the page application called `PAGENETF` that is normally found on Page F of the "little consoles." One can view UDP datagrams, ICMP and IGMP packets, IP fragments, and ARP messages. (This data stream support also exists for PowerPC nodes, and one can view UDP datagrams in this way.) Here is an example of network activity captured from `node0600`, the Linac server node, through which nearly all Acnet requests for Linac data pass.

```
SrcN Size T ^Frame HrMn Sc Cy+ms
A62F 0016 R E6D27A 1359:41-00+13  15 Hz data from contributing nodes
A62D 0060 R E6D2A2 1359:41-00+14  in response to active periodic requests
A611 0016 R E6D316 1359:41-00+14  etc
A610 001A R E6D33E 1359:41-00+14
A625 006E R E6D36A 1359:41-00+14
A622 006E R E6D3EA 1359:41-00+14
A623 006E R E6D46A 1359:41-00+14
A624 006E R E6D4EA 1359:41-00+14
A626 006E R E6D56A 1359:41-00+15
A621 006E R E6D5EA 1359:41-00+16
A627 006E R E6D66A 1359:41-00+16
A620 00A4 R E6D6EA 1359:41-00+17
A92E 0208 R E6D7A2 1359:41-00+17  New RETDAT request from cns46
A9FC 0208 T E935D4 1359:41-00+18  Multicast to all Linac nodes
A600 0208 R E6D9BE 1359:41-00+18  Receive own multicast
A613 0016 R E6DBDA 1359:41-00+19  First of 15 contributing node replies
A611 001A R E6DC02 1359:41-00+20  etc
A614 0016 R E6DC2E 1359:41-00+20
A612 0016 R E6DC56 1359:41-00+20
A615 003E R E6DC7E 1359:41-00+20
A622 0016 R E6DCCE 1359:41-00+20
A625 0016 R E6DCF6 1359:41-00+21
A627 0016 R E6DD1E 1359:41-00+21
A623 0016 R E6DD46 1359:41-00+21
A626 0016 R E6DD6E 1359:41-00+21
A624 0016 R E6DD96 1359:41-00+21
A61C 009C R E6DDBE 1359:41-00+21
A621 001A R E6DE6E 1359:41-00+21
A62D 001A R E6DE9A 1359:41-00+21
A61E 0016 R E6DEC6 1359:41-00+21  Last contributing node reply
A92E 008E T E937EE 1359:41-00+21  Return composite reply to cns46
A921 003C T E9388E 1359:41-00+48  Usual periodic replies
A9E8 0022 T E938DC 1359:41-00+48  etc
```

Each line represents a datagram that was either transmitted or received, and the node is specified in terms indicating a native node number, with the first digit `A` representing the Acnet protocol port. When a native node number does not apply, the Acnet node number is specified, such as `92E` (trunk 9, node 46) for `cns46`. The apparent one-shot request was turned around in this example in only 4 ms from the time `node0600` received the request from `cns46`, in spite of the fact that it had to gather data from 15 other front ends before it could deliver the reply. (The times are shown in `HrMn:Sc-Cy+ms` format, indicating time within a particular 15 Hz accelerator cycle down to millisecond resolution.)

Interrupt timing

Recent diagnostics added support measuring particular interrupt activities, including ethernet transmit and receive activitiess separated out. The interrupt timing table (`ITT`) is located at low memory address `0x800`, with a total length of 512 bytes. A 32-byte header is followed by as many as thirty 16-byte entries. The entries have the following format:

Field	*Size*	*Meaning*
id	1	Identifier for this entry
flgBit	1	Flags in hi nibble, LED bit# in low nibble.
count	2	Activity counter
elapsed	4	Elapsed time
base	4	Base time for measuring elapsed time
cycle	4	Time of interrupt within current 15 Hz cycle.

Here is an example of what the ethernet entries look like, captured at random:

```
000860      E000 9A82 0000 0013     Ethernet Output
000868      9EDC C237 0000 0AA0

000870      E100 1F76 0000 0039     Ethernet Input
000878      9EAC 9150 0000 AA31

000910      EC83 BA04 0000 002F     Ethernet Complete
000918      9EDC C222 0000 0A8B
```

Watching this area of memory gives a rough idea of ethernet interrupt activity, including the elapsed time to perform it. All timings are in microseconds. The last one indicates that interrupt LED #3 is used for exhibiting Ethernet Complete activity, which is the entire duration of the ethernet interrupt routine. (The Ethernet Output and Ethernet Input measure only the time of Output or Input processing, respectively, within the interrupt routine.)

One can perform related timing tests by modifying the flags nibble. As mentioned already, the flags value of 8 specifies using a LED to make the activity externally visible. A value of 4 alters the elapsed time to measure the time between successive activities. The value 2 specifies keeping the maximum duration in the elapsed time field (or the minimum time between successive activities if the flag value of 4 is also set). The value 1 specifies that the elapsed time records should also be logged in the data stream called `ITTLOG`.

Ethernet Interrupt Times

This is an older version of ethernet interrupt timing that is also available. It is located at low memory address `0x4C00` and has a size of 1024 bytes. Each entry is 8 bytes long and

uses the following format:

Field	*Size*	*Meaning*
`status`	2	Status word from System Command Block
`cycCnt`	2	Cycle counter, low word of `CYCLECNT` global variable
`cycTime`	2	Time in present 15 Hz cycle, in 0.5 ms units
`elapsed`	2	Elapsed time in interrupt routine, in microseconds

Here is an example showing entries that resulted when a device name lookup was initiated:

```
:00004EC8   6040 B1D6 0007 0074  Multicast name query
:00004ED0   4040 B1D6 001E 0044  Response from node0611
:00004ED8   6040 B1D7 000A 0070  ARP request
:00004EE0   4040 B1D7 000B 0040  ARP reply
:00004EE8   2040 B1D7 000D 0033  Device data request
:00004EF0   4040 B1D7 001D 0045  Device data replies
:00004EF8   4040 B1D8 001B 004A  etc.
:00004F00   4040 B1D9 001B 004E
```

The upper 4-bit nibble of the status word indicates that the interrupt was due to a received frame (4), a transmitted frame (2), or both (6). The second nibble indicates the status of the Command unit in the ethernet chip (intel 82596), which includes transmit commands; a value of 0 means the Command unit is idle. The third nibble indicates the status of the Receive unit; a value of 4 indicates that the Receive unit is ready. The name lookup query is indicated in the first entry shown, where the node sending the query also received its own multicast query message. The second line shows the response from the node having the named device. The third line shows the ARP request, which is followed by the ARP reply. The fifth line shows the query for device data as initiated by the parameter page application. This is followed by the immediate first reply, then by additional replies occurring at a 15 Hz rate.

Ethernet Receive Unit Not Ready

As a result of an early experience encountering a condition of Not Ready status for the Receive unit, caused at that time by receiving a frame more than 1500 bytes in length, this diagnostic was added in case it occurs again. This table is based at `0x6000` and is 1024 bytes in size. The first 8 bytes of each 64-byte entry hold the usual BCD time-of-day, followed by the 16-byte RFD (Receive Frame Descriptor), followed by the first 40 bytes of the received frame. The system restarts the Receive unit if this condition is detected. For further details, refer to the `ERUREADY` code in the Ethernet.a source module.

Ethernet Receive timing

This diagnostic focuses on the number of receive frames processed within one ethernet interrupt activity. It is located at `0x6400`, with a size of 1024 bytes. An 8-byte header includes the current offset word to the next entry, a byte containing the maximum number of frames processed during one interrupt, and 5 bytes that show the time when this maximum count was last increased, specified in `Mo-Da-Hr-Mn-Sc`. Each 4-byte entry consists of a byte count of frames received in one interrupt, and a 3-byte microsecond elapsed time since the previous such receive frame interrupt. Here is an example of such data:

```
:000064E0   01 0509E4  01 02D9D8
:000064E8   01 00A20A  01 021BED
:000064F0   02 05F5D5  00 000080
:000064F8   01 09C5A9  01 06391B
```

```
:00006500   01 06A2FC  01 0270C0
:00006508   01 033319  01 02D8C3
:00006510   01 13C9EA  01 005DF0
:00006518   01 037876  01 00571F
```

Most of the time, only one received frame is processed per interrupt. But one entry shown here indicates that two frames were proceed in one interrupt, followed by an entry that indicates that no receive frames were processed. This may result from the second frame arriving while the first frame was being processed. With the interrupt having been cleared early in the interrupt routine, the ethernet chip caused another interrupt, but by the time the interrupt routine finished processing both frames, there was nothing more to do.

Foreign Node Capture

Another diagnostic captures unusual accesses via IP, which means here at Fermilab that a node sent an IP datagram from outside the lab. In this case, "outside the lab" indicates that the source IP address is something other than `131.225.x.x`. Recently, entries were seen in this table, located at `0x6800` with size 2048 bytes, in which the source IP address was actually a multicast IP address, which is clearly invalid. The captured part of the frame showed these datagrams to be ICMP "port unreachable" messages that were apparently sent by some unsophisticated nodes embedded in commercial equipment. The idea of this diagnostic is to record the kinds of accesses made by nodes outside our lab. The records in this table are 128 bytes long, with the first 8 bytes being the time-of-day of a first access, followed by the first 56 bytes of the initial IP datagram (beginning with the IP header), followed by a count of the number of accesses (using a 30-second timeout), followed by a timeout counter, followed by the 5-byte time-of-day of the final access, as `Hr-Mn-Sc-Cy-00`, finally followed by up to 56 bytes of the final IP datagram received. Here is an example:

```
0602:006980 0209 2312 3417 0900  first access at 09/23/02 1234:17
0602:006988 4500 0025 F517 0000  first IP header, datagram
0602:006990 7001 2855 420D E3AE  IP address 66.13.227.174
0602:006998 83E1 83CE 0800 CE5C  ICMP ping request
0602:0069A0 0300 6471 6865 6C6C
0602:0069A8 6F20 3F3F 3F00 0000
0602:0069B0 0000 0000 0000 00F0
0602:0069B8 003C 0000 0E4E 0C00

0602:0069C0 0002 0012 3422 0000  2 accesses, final at 1234:22
0602:0069C8 4500 0025 F5A7 0000  final IP header, datagram
0602:0069D0 7001 27C5 420D E3AE  ICMP ping request
0602:0069D8 83E1 83CE 0800 6F5C
0602:0069E0 0300 C371 6865 6C6C
0602:0069E8 6F20 3F3F 3F00 0000
0602:0069F0 0000 0000 0000 00D4
0602:0069F8 0035 0000 0E4D 0C00
```

Ethernet variables

In addition to the diagnostic tables described above, there are a number of variables maintained by the system code that relate to ethernet communications. First are the static variables maintained by the ethernet driver. These are located at `0x160000`, of size 160 bytes. For easy correlation with the Memory Dump page, the fields are separated every 8 bytes, and also every 64-byte page.

Offs	*Field*	*Size*	*Meaning*
00	SCBSTAT	2	Status word from 82596
	SCBCMND	2	Command word for 82596
	SCBCPTR	4	(ws) ptr to command block list
08	SCBRPTR	4	(ws) ptr to receive frame area
	SCBNCRC	4	#CRC errors
10	SCBALGN	4	#alignment errors
	SCBRSRC	4	#resource errors
18	SCBOVRN	4	#overrun errors
	SCBCOLL	4	#collisions
20	SCBSHRT	4	#short frames
	SCBTOFF	2	t_off
	SCBTON	2	t_on
28	SCBNOTC	4	CFN interrupt without C bit set
	RCVBADF	4	Bad frame received
30	RCVBADC	4	No F bit set in RFD
	RCVBADL	4	Bad length received
38	SCBCINT	4	#Command interrupts
	SCBRINT	4	#Receive interrupts.
*	----------		
40	RCVINVM	6	Invalid multicast address
	RCVINVC	2	Invalid multicast address counter
48	SAPQERR	2	Error from Send_X call
	SAPQAUX	2	Error status code
	UIZSAP	2	Count uninitialized SAP received
	UIZSAUX	2	SAP value
50	BADSAP	2	Count unknown SAP value
	BADSAUX	2	Unknown SAP
	SPARERR	2	spare
	SPARAUX	2	spare
58	SRCADRC	2	Count invalid source address
	SRCADRA	6	Record invalid network address
60	ERCVCUR	4	Ptr to next RFD to be processed
	SPURINT	4	Count of spurious interrupts
68	EINTCNT	4	Count of interrupts (total)
	EBUSCNT	4	Count of bus error interrupts
70	EBUSCOD	1	Error code byte from status register
		3	spare
	XMTCOLL	4	Collisions from status word in transmit command

```
78    XMTSTAT     2     Transmit status word last collision
      XMTADDR     6     Destination address last collision
*     ----------------------
80    EITIMEO     2     Offset into interrupt times diagnostic array
      ERUDCNT     2     Delay after re-init Receive Unit
      ERUDIN      2     Offset to next entry in Receive Unit diagnostics
      ERUDTOT     2     Receive Unit not ready status total count

88    ESPARE     24     spare
A0    (end)
```

Here is an example that illustrates some of these variables:

```
      0509:160000 0040 0000 B5C0 0019  word-swapped ptr to command block list
      0509:160008 0200 0016 0000 0000  word-swapped ptr to receive frame area
      0509:160010 0000 0000 0000 0000
      0509:160018 0000 0000 0000 0000
      0509:160020 0000 0000 0019 00AF
      0509:160028 0000 0000 0000 0000
      0509:160030 0000 0000 0000 0000
      0509:160038 0000 ADE7 001B DA41  #command ints, #receive ints.

      0509:160040 3333 FF98 99A6 0008  invalid multicast address, counter
      0509:160048 0000 0000 0000 0000
      0509:160050 6BE2 00E0 0000 0000  unknown SAP counter, value
      0509:160058 0000 0000 0000 0000
      0509:160060 0016 5600 0000 0128  ptr to next RFD to process, spurious cnt
      0509:160068 001C 82EE 0000 0000  #interrupts (total)
      0509:160070 0000 0000 0000 0018  #collisions from status word in xmit cmnd
      0509:160078 2001 0030 7B92 C170  status word last collision, net address

      0509:160080 01D8 0000 0000 0000  offset into EITimes array
      0509:160088 0000 0000 0000 0000
      0509:160090 0000 0000 0000 0000
      0509:160098 0000 0000 0000 0000
```

SNAP *variables*

The SNAP task supports IP protocols, and it uses a number of variables. Here is the layout of this structure:

```
Offs  Field    Size     Meaning
FFD4  FNCAPO      2     Foreign node capture buffer offset to next record
      FNCAPTC     2     Foreign node capture timer countdown period

FFD8  FNCAPNT     4     Foreign node capture total number of records written
      ARPREQI     4     ARP request ignored counter

FFE0  ARPREQC     4     ARP request for this node counter
      ARPRPYC     4     ARP replies counter

FFE8  ICMPCNT     4     ICMP received counter
      IPSNAPC     4     IP using SNAP on ethernet counter
```

```
FFF0  IPSNAPA    4      IP using SNAP on ethernet source IP address
      ARPSNAPC   4      ARP using SNAP on ethernet counter

FFF8  ARPSNAPA   4      ARP using SNAP on ethernet source IP address
      NETFDSX    2      Data stream index for net frame diagnostics
      NETFDSS    2      Error return status from DSWrite call
*     --------------------------
* Frame message queue buffer (6 longwords)
0000  FMRSVD     4      reserved
      FMHOME     4      home xid

0008  FMSIZE     2      Frame contents size
      FMMSGC     2      Frame message count
      FMSOURCE   2      Offset to source address
      FMDEST     2      Offset to destination address

0010  FMPTR      4      Ptr to frame contents
      FMSRCN     2      Pseudo source node#
      FMIPOFF    2      Offset to IP header
*
0018  SNAPXID    4      SNAP queue id
      SNAPCREE   1      Error return from Create_X call
      ETHERF     1      Ethernet flag
      SNAPREQE   2      Error return from Req_X call

0020  BADOERL    4      Unknown organization id
      BADOERC    2
0026  SRCNETA    6      Source network address
      IPHLNG     2      IP header length
002E  BADIPERR   4      Invalid destination IP address
      BADIPERC   2
0034  FMSIZERR   2      Bad frame size
      FMSIZERC   2
0038  UPEVERR    2      UDP Signal_V error code
      UPEVERC    2
      UPSQERR    2      UDP Send_X error
      UPSQERC    2
*     --------------------------
0040  UPCKERR    2      UDP checksum error
      UPCKERC    2
      BADPERR    2      Unknown protocol type
      BADPERC    2
0048  UIZPERR    2      Uninitialized port#
      UIZPERC    2
      TCPPERR    2      TCP unsupported yet
      TCPPERC    2
0050  UNKPERR    2      Unexpected protocols besides TCP
      UNKPERC    2
      TTLERR     2      Invalid TTL value
      TTLERC     2
0058  IPFOERR    2      IP fragment offset nonzero
      IPFOERC    2
      IPFLERR    2      IP fragment length invalid
      IPFLERC    2
```

```
0060  IPCKERR    2      IP checksum error
      IPCKERC    2
      IPVNERR    2      IP version# error
      IPVNERC    2
0068  ARPOERR    2      Unsupported ARP operation
      ARPOERC    2
      ARPLERR    2      Unexpected address lengths
      ARPLERC    2
0070  ARPPERR    2      Unexpected ARP protocol type
      ARPPERC    2
      ARPHERR    2      Unexpected hardware type
      ARPHERC    2
0078  ICCKERR    2      ICMP checksum error
      ICCKERC    2
      ICPUERR    2      Port unreachable error
      ICPUERC    2
*     --------------------------
0080  ICTCERR    2      ICMP type/code not supported
      ICTCERC    2
      ICEVERR    2      ICMP Signal_V error code
      ICEVERC    2
0088  ICSQERR    2      ICMP Send_X error status
      ICSQERC    2
      IGCKERR    2      IGMP checksum error
      IGCKERC    2
0090  LAPROERR   2      Local appl for UDP port inaccessible
      LAPROERC   2
      DSTPORTN   4      Dest UDP port# as taskName
0098  DATAGRAM   4      Ptr to completed datagram block from FrgIPARP.
```

Here is an example of these variables:

```
:0004CE10   xxxx xxxx xxxx xxxx
:0004CE18   xxxx xxxx xxxx xxxx
:0004CE20   xxxx xxxx 0780 001E  Foreign node capture offset, countdown.
:0004CE28   0000 005F 000C FEA6  Foreign node capture count, #ARPs ignored
:0004CE30   0000 0002 0000 00ED  #ARP requests for this node, #ARP replies
:0004CE38   0000 003D 0000 0000  ICMP received count 003D
:0004CE40   0000 0000 0000 0000
:0004CE48   0000 0000 0000 0000

:0004CE50   0000 0000 0000 0000  FMRSVD, FMHOME
:0004CE58   0020 FFE4 FFFC FFF6  FMSIZE, FMMSGC, FMSOURCE, FMDEST
:0004CE60   0017 941A 0000 0000  FMPTR, FMSRCN, FMIPOFF
:0004CE68   0002 05FA 00FF 0000  SNAP queue id
:0004CE70   0000 0000 0000 0240  source net address 0240 0000 0593
:0004CE78   0000 0593 0014 83E1  IP hdr lng, invalid dest IP 83E17DFF
:0004CE80   7DFF 1D52 0000 0000  invalid dest count = 1D52
:0004CE88   0000 0000 0000 0000

:0004CE90   0000 0000 8137 1986  unknown protocol type 8137, count
:0004CE98   0043 B85A 0006 05B1  unknown port 43, count, TCP, count
:0004CEA0   0000 0000 0000 0000
:0004CEA8   0000 001B 0000 0000  invalid TTL value, count
```

```
     :0004CEB0   0000 0000 0000 0000
     :0004CEB8   0000 0000 0000 0000
     :0004CEC0   0000 0000 0000 0000
     :0004CEC8   0000 0000 0000 0000

     :0004CED0   0A00 0003 0000 0000  unknown ICMP type/code 0A00, count
     :0004CED8   0000 0000 0000 0000
     :0004CEE0   0000 0000 0011 1A90  dest UDP port# as taskname 00111A90
     :0004CEE8   0003 9C78 0000 0000
```

Broadcast diagnostics

Another class of diagnostics captures records of several kinds of broadcasts. Each kind is selected by storing a key word value in the first word of the memory used for this purpose at `0x5000`, with size 1024 bytes. Besides the 2-byte keyword value, the header includes a 2-byte count of active entries, and a 4-byte total count. Each entry following the header is 8 bytes in length, the format of which depends on the type of diagnostics chosen. Each type is described in turn:

`Type 'AR' = 0x4152`

This type monitors ARP requests received. Each 8-byte record indicates the sender's IP address, the operation code (1=ARP, 3=RARP), and a counter.

`Type '11' = 0x3131`

This type monitors ARP requests from 131.225.1.1 specifically. Each record indicates the sender's IP address, the ARP/RARP operation code, and a counter. (This type was installed long ago when a network problem occurred that presumably no longer exists.)

`Type 'BC' = 0x4243`

This type monitors non-IP broadcasts. The 8-byte record format includes the 6-byte network address and a counter.

`Type 'MC' = 0x4D43`

This type monitors datagrams received via multicast that are not registered in the multicast table at TRING+`0x240`, or `0x405240`. Each entry includes the 6-byte multicast network address followed by a counter.

Here is an example of this kind of diagnostic, with the keyword set to **`'AR' = 0x4152`** to monitor ARP request activity.

```
 0509:00005000   4152 002C 0000 021E  2C active entries, total count = 21E
     :00005008   83E1 88C8 0001 0048  popular ARP requester .136.200
     :00005010   83E1 7B34 0001 0021
     :00005018   83E1 8896 0001 0020
     :00005020   83E1 8801 0001 004C  popular ARP requester .136.1
     :00005028   83E1 887F 0001 0001
     :00005030   83E1 7B97 0001 002A  popular ARP requester .123.151
     :00005038   83E1 8872 0001 0002
     :00005040   83E1 7B08 0001 0001
     :00005048   83E1 7BC8 0001 0015
     :00005050   83E1 7B01 0001 0017
     :00005058   83E1 10A8 0001 0014
     :00005060   83E1 7D83 0001 0003
```

```
:00005068   83E1 10C8 0001 0093  popular ARP requester .16.200
:00005070   83E1 88FD 0001 0002
:00005078   83E1 7B80 0001 0003
:00005080   83E1 7DC8 0001 0011
:00005088   83E1 1089 0001 0004
:00005090   83E1 7D93 0001 0002
:00005098   83E1 1083 0001 0006
:000050A0   83E1 7BF2 0001 0001
:000050A8   83E1 7DD7 0001 0001
:000050B0   83E1 7DBB 0001 0002
:000050B8   83E1 7D7D 0001 0002
:000050C0   83E1 104B 0001 0001
:000050C8   83E1 8878 0001 0005
:000050D0   83E1 105B 0001 0001
:000050D8   83E1 1008 0001 0001
:000050E0   83E1 7B2F 0001 0001
:000050E8   83E1 7BF5 0001 0001
:000050F0   83E1 883D 0001 0002
:000050F8   83E1 7B43 0001 0002
```

The ARP requesters pointed out above are

```
131.225.136.200   vlan3.r-ad-6509-xgc108-1.fnal.gov
131.225.16.200    vlan3.r-ad-6509-xgc108-1.fnal.gov (same)
131.225.136.1     beamssrv1.fnal.gov
131.225.123.151   beams-pdc.fnal.gov
```

The first two are apparently a router used by Beams division. It is of course natural that a router should make lots of ARP requests.

Frame Monitor local application

There is a local appplication called FMON that can be used to capture selected network frame activity. Its parameters specify an area of memory to use for a circular buffer, which is often chosen to be based at 0x300000, using the 4th megabyte of an IRM's 4 MB memory. A target node number is also specified, and one can elect to capture frames being sent from the local node to the target node, frames received by the local node from the target node, or both. In addition, one specifies a word offset from the start of the frame, because the amount of data captured is limited. Each 64-byte record starts with the usual 8-byte time-of-day, followed by up to 56 bytes of frame contents, beginning at the specified word offset. If the offset is 7, for example, it means that the 14-byte ethernet frame header is skipped. If the offset is 17, or 0x0011, then both the frame header and the IP header are skipped, so that the captured frame contents begin with the 8-byte UDP header. The time-of-day fields show time down to the half millisecond of the given 15 Hz cycle. Since IP node numbers are pseudo node numbers, one may need to consult the contents of the IPARP table in order to come up with the pseudo node number of interest. To make this step easier for capturing Acnet protocol frames, one can specify an Acnet node number, and the pseudo node number will be worked out automatically.

RETDAT ***Log***

Acnet RETDAT requests are logged in a data stream called RETDLOG, if it is defined inside a node. The 16-byte entries are comprised of the following fields:

Field	*Size*	*Meaning*
node	2	node number from which request was received
bytes	2	#reply bytes expected, including 2-byte status word(s)
dev	2	#devices in request
ftd	2	frequency-time descriptor
msgId	2	message id from Acnet header
time	6	time-of-day request received, as Da-Hr-Mn-Sc-Cy-ms

The main purpose of this diagnostic is to show the timing of RETDAT requests and what kind of requests they are. (This diagnostic also exists in the PowerPC nodes.) For example, an ftd value of 0000 means a one-shot request, which is very common. An ftd value of 003C would be used for a request that asks for 1-second replies. A cancel message is recorded as a special record in which the bytes, dev, and ftd are all 0000. Use the msgId field to associate a cancel with a prior request.

One can apply filtering to the logging in order to select requests from certain nodes. To do this, find the "user area" portion of the data stream queue header, which consists of 4 words that can be set to Acnet node numbers of interest. By default after system initialization, all four words are zero. But if the first word is set nonzero, it means that the *inclusive* mode of filtering is used, so that a request from any of up to 4 nodes specified will be logged, while requests from nodes not specified will be ignored. If the first word is zero, but one or more of the other words are nonzero, then the *exclusive* mode of filtering is used, so that a requesting node matching any of up to 3 nodes specified will be ignored, and a request from any node not listed will be logged. The data stream queue is usually located at 0x9000, with a size of 4096 bytes, so that the user area is found at 0x9018. Here is an example of some of such records as sampled from node0600, which receives lots of Acnet requests.

```
:00009560   092E 00F0 003C 0000 6402 0313 1501 0030
:00009570   0901 0010 0004 0000 440C 0313 1501 0030
:00009580   0908 0004 0001 0000 4003 0313 1501 0865
:00009590   0905 0010 0004 0000 A802 0313 1502 024F
:000095A0   0B21 0028 000A 0000 AB00 0313 1503 0433
:000095B0   091E 0004 0001 0000 C40F 0313 1504 1435
:000095C0   0908 0004 0001 0000 3403 0313 1505 1463
:000095D0   090C 0004 0001 0000 8001 0313 1509 0031
:000095E0   091C 0014 0005 0000 B002 0313 1511 0004
:000095F0   092E 007C 001F 0000 6002 0313 1511 0024
:00009600   09E8 0004 0001 003C 942D 0313 1511 1284  <--
:00009610   0908 0004 0001 0000 2003 0313 1512 0864
:00009620   090D 0004 0001 0000 0404 0313 1515 0024
:00009630   09E8 0000 0000 0000 942D 0313 1515 0455  <--
```

During the 15 seconds of RETDAT request activity logged here, most entries seem to be one-shot requests, likely initiated by data logger clients. This brief snapshot does show a 1-second request (using msgId = 942D) from node 09E8, which is cfss, that was canceled about 3.5 seconds later.

Task activity

Although not limited to ethernet diagnostics, a data stream called TASKLOG is often implemented in front ends. In a similar way that the NETFRAME data stream captures network datagram activity, as described early in this document, TASKLOG captures all task activity in a

system. (This is also supported for PowerPC nodes.) Here is an example of task activity featuring some network-related support.

```
Task   Evnt   ETim HrMn:Sc-Cy+ms
QMon   0000    .11 1721:55-10+ 0
Updt   0000   4.17 1721:55-10+ 0  Normal data pool update, including BLMS
QMon   0001    .21 1721:55-10+ 4
DTim   0000    .07 1721:55-10+ 5
Alrm   0000    .16 1721:55-10+ 5  Alarm scanning
Appl   0000    .29 1721:55-10+ 5  Page application
Updt   0000    .1  1721:55-10+ 5
QMon   0000    .06 1721:55-10+ 5
IDLE   0000  25.33 1721:55-10+ 5
SNAP   0000    .32 1721:55-10+31  Request arrival, IP processing
ANet   0000    .13 1721:55-10+31  Acnet dispatching
ACRq   0000    .54 1721:55-10+31  RETDAT handling
Updt   0000   1.1  1721:55-10+32  Build reply
QMon   0000    .18 1721:55-10+33  Clean up
IDLE   0000    .3  1721:55-10+33
SNAP   0000    .9  1721:55-10+33  (unrelated request arrival)
ANet   0000    .15 1721:55-10+34  (Acnet dispatching)
ACRq   0000    .23 1721:55-10+34  (RETDAT handling, ignored)
IDLE   0000   5.68 1721:55-10+34
Cons   0000    .07 1721:55-10+40
Serv   0000    .13 1721:55-10+40
Updt   0000    .11 1721:55-10+40
QMon   0000    .25 1721:55-10+40
IDLE   0000  26.09 1721:55-10+41

SrcN Size T ^Frame HrMn:Sc-Cy+ms
A562 0040 R 165C22 1721:55-10+31  Request received
80FF 0018 T 18E90A 1721:55-10+32  Reply ready, but ARP request first
A6CF 0032 R 166822 1721:55-10+33  (unrelated request received)
0562 0018 R 166E22 1721:55-10+33  ARP response
A562 042C T 18E93C 1721:55-10+34  Send reply
```

For this example, a request for 1024 bytes of waveform data from a Booster beam charge signal was issued as a one-shot request. Both the task level processing and the corresponding network diagnostics are indicated in support of this request. At the task level, the request was handled in 2.27 ms, although at the network level, we see 3 ms, in part due to the test request being a surprise, so that an ARP request was needed to obtain the 6-byte network address of the requesting node in order to deliver the reply. An unrelated request was received from the Booster server node, which was ignored, since we see no further task activity resulting from its `RETDAT` processing. From the task level processing, we can see that in this same 15 Hz cycle, the CPU was only 14% busy.

References

A number of other documents relate to subjects covered in this one. For detail on the ethernet chip used in the MVME-162 boards, see the intel document “32-bit LAN Component User’s Manual.” Other documents concerning particular diagnostic support are:

Network Frames Page, *Built-in network diagnostics*

Source code documents, of course, are behind all IRM code execution.

System code modules:

Name	*Support*
`Ethernet.a`	Ethernet driver, interrupts, diagnostics
`ITTStart.a`	Interrupt timing table
`Switch.a`	Task switch hook
`SNAP.a`	Internet protocols
`ACReq.a`	`RETDAT`
`FMON.p`	Frame Monitor capture (local application)
`NETF.p`	Network frames listing (Page F)
`TASK.p`	Task activity listing (Page ?)
`PMEM.p`	Print memory (Page H)
`MDMP.p`	Display memory contents (Page M)

For the PowerPC system support, several similar modules exist, as follows:

Name	*Support*
`ITTStart.c`	Interrupt timing table
`TrigTask.c`	Task switch hook, `vmeSwitchHook()`
`socketReadControl.c`	Datagram diagnostic logging, calls `recvfrom()`
`socketWriteControl.c`	Datagram diagnostic logging, calls `sendto()`
`foreignNodeCapture.c`	Foreign node capture logging
`ACReq.c`	`RETDAT`
`LOOPFMON.c`	Frame Monitor capture
`PAGENETF.c`	Network frames listing (Page F)
`PAGETASK.c`	Task activity listing
`PAGEMDMP.c`	Display memory contents