ETHERNET PERFORMANCE ANALYSIS

by

Alex Colvin
Thomas Donnelly
Robert Simoncic
Alfred C. Weaver

Computer Science Report No. RM-86-01
March 14, 1986

ETHERNET PERFORMANCE ANALYSIS

Measurements of File Transfers, Virtual Circuits, and Datagrams

at the Application, Session, Transport, and Data Link Layers

Using Intel iSBC 552 Ethernet Communications Controllers

and iNA 961 Networking Software

Alex Colvin
Thomas Donnelly
Robert Simoncic
Alfred C. Weaver

March 1986

TABLE OF CONTENTS

Chapter 1

Introduction

1.1. Scope

This report describes the methods and summarizes the results of a series of performance measurements conducted on an Ethernet Local Area Network. Experiments on two different operating systems (XENIX 286 and iRMX 86) and their associated networking facilities were performed; throughput, collision rates, and delay times were measured for various configurations of hardware and offered load.

1.2. Physical Description

For this series of experiments three Intel System 310 microcomputers were linked in an Ethernet through cables connected to an iDCM 911-1 Intellink Cluster Module. The microcomputers were configured with 80286 microprocessors and iSBC 552 Ethernet Communications Controllers. Two systems, Nodes 1 and 3, had 36-megabyte Quantum hard disks and one, Node 2, had a 19-megabyte hard disk.

Chapter 2

XENIX 286 Measurements

2.1. Software

Software for this set of measurements consisted of the XENIX 286 3.0 Operating System operating through XENIX Networking Software. The iNA 961 software package provided implementation of the ISO transport layer (layer 4) and logical link control sublayer (upper half of layer 2). (ISO layer 3 is null in Ethernet.)

2.2. Description of Experiment

Because the XENIX software precludes direct access to layers 4 and below, only the performance characteristics of ISO layers 7 and 5 were measured (layer 6 is null).

For measurements at layer 7, bulk file transfers were invoked through the XENIX Networking File Consumer and File Server Software, whereas experiments on layer 5 were conducted by means of the Session Interface. All events were timed by the XENIX "time" command.

Files of approximately one-half and one megabyte, respectively, were transferred among various combinations of sender-receiver pairs (virtual circuits) by means of the XENIX "move" command for the layer 7 measurements.

For the one-megabyte case identical files were transferred between nodes. Due to disk congestion it was not possible to increase the number of virtual circuits beyond two. For the 1/2-megabyte case identical files were again transferred

between nodes. Averaged results are summarized for several combinations of virtual circuits and number of nodes interacting. Disk congestion again precluded increasing the complexity beyond two virtual circuits on each of the three nodes.

For the measurements at layer 5, messages of 512 bytes and 1024 bytes were transferred through various numbers of virtual circuits and nodes. In order to achieve reasonable timing accuracy these messages were sent 10000 or 20000 times, with the total time being used to compute the average throughput.

2.3. Layer 7 Results

Results of the file transfer rate measurements are summarized in Table 2-1. The number of virtual circuits (sender-receiver pairs) is varied for two file sizes, and transfer rates are shown in both bytes per second and bits per second.

File Transfer Rates			
Virtual Circuits	Size (Kilobytes)	Bytes/sec	Bits/sec
1	525	17773	142184
2	"	10766	86128
4	"	6570	52560
6	"	6409	51272
1	1024	17705	14160
2	"	10744	85952

Table 2-1.

File Transfer Rates.

As previously mentioned, it was not possible to increase the number of virtual circuits beyond two for the one-megabyte transfers, or beyond six for the

1/2-megabyte transfers. This was presumably due to lack of disk space. The overhead required for transfer appears to be substantial, for the amount of free disk space was significantly larger than the files to be moved in all cases.

The transfer times presented are averages. Individual times actually varied greatly, being larger by a factor of up to 1.67 for transfers from a less- to a more-crowded disk (usually the 19-megabyte system) than for transfers from a more- to a less- crowded disk.

The decrease in efficiency observed with increasing number of virtual circuits was surprising, and it is to be noted that the maximum throughput was only 1.4% of the Ethernet's rated capacity of 10 megabits per second.

2.4. Layer 5 Results

Results for the messsage-transfer measurements are summarized in Table 2-2, in which virtual circuits per node are varied for two packet sizes and three combinations of transmitting nodes, with results given in bits per second.

Message Transfer Rates				
Virtual Circuits	Size (Bytes)	1 Node Sending (Bits/Sec)	2 Nodes Sending (Bits/Sec)	3 Nodes Sending (Bits/Sec)
1	512	158144		
2	"	250904	225672	
3	"	295856		
4	"	295208	256952	
5	"	294680		
6	"		265040	
8	"		270088	
10	"		287936	
1	1024	283752		
2	"	383080	459176	
3	"	481688		
4	"	530088	482280	
5	"	533664		
6	"		476128	742568
8	"		487384	
10	"		517736	
12	"			760912
18	"			786112

Table 2-2.

Message Transfer Rates, Several Nodes and Virtual Circuits.

As expected, message lengths of 1024 bytes gave almost double the performance of 512-byte messages, since the packet buffer on the iSBC 552 board has a

capacity of 1024 bytes.

In general throughput increased with the number of virtual circuits, which reflects the fact that the hand-shaking overhead was being partially masked. The slight decline observed in throughput after three virtual circuits in the one-node case may simply mean that saturation had been reached and further increases in circuits merely added to the handshaking overhead. On the other hand it is not immediately obvious what caused the throughput decline at six virtual circuits for the 1024-byte two-node sending case. Moreover performance usually declined going from one to two nodes sending.

Performance was not measured beyond 18 virtual circuits because of a limit of 12 (six senders, six receivers) virtual circuits per node imposed by the XENIX kernel configuration.

Chapter 3

iRMX 86 Measurements

3.1. Software

Software for this phase of measurements consisted of the iRMX 86 Operating System, Release 6, operating through iRMX Networking Software, and the iNA 961 software package.

3.2. Description of Experiment

3.2.1. Layer 7

Files of various sizes were transferred among different combinations of nodes, and times were measured through the system "time" command. Since calls to "time" must act through the I/O system, causing a significant overstatement of total delay, this I/O overhead was measured and subtracted from the results to find the true duration of transfer delay.

Files of 32k bytes to 8M ASCII bytes (for some measurements 8k- and 16k-byte files were also tested), in increments of powers of two, were used. Lack of disk space on the 19-megabyte system precluded measuring 8-megabyte transfer rates in some cases.

The experiment was divided into six parts, as follows :

1. One-way transfers, Node 1 to Node 2.
2. Simultaneous two-way transfers, Node 1 to Node 2, and Node 2 to Node 1.
3. Simultaneous two-to-one transfer, using Node 2 to Node 1, and Node 3 to Node 1.

4. Three-way (cyclic) transfer, using all nodes.

5. Simultaneous two-to-one and one-to-one transfer, using Nodes 2 to 1, 3 to 1, and 1 to 3.

6. Internal file transfer. This was done on Node 1, as a check to estimate how much of the extenal transfer time was due to the characteristics of the file system itself.

Because of granularity of the timer, each file was transferred at least 10 times to provide reasonable accuracy (this was not always true for eight-megabyte files, where the long times involved allowed sufficient accuracy).

3.2.2. Layers 4 and 2

For this set of experiments timing information was derived from the 8254 Programmable Interval Timer. It should be noted that calls to the timer require 0.36 milliseconds to execute, and that therefore there is an uncertainty of that amount in all time measurements.

The iNA Transport Service implements layer 4 and provides two types of service: a reliable connection-oriented virtual circuit (VC) message delivery service, and a non-guaranteed connectionless datagram message delivery service. Layer 2, the data link layer, is responsible for transmitting data over the network, and performs framing, addressing, and collision handling. This layer is a datagram service only, and does not ensure accurate reception of data.

Experiments at layers 4 and 2 measured three distinct delay times: the time required for submitting a request block, the time required for a mailbox response, and the total time required for message delivery. Message sizes of 32, 128, 512, and 1500 bytes were used, and for each size statistics were kept for the various delays.

Three groups of experiments were performed. First, comparisons were made between the layer 4 virtual circuit, layer 4 datagram, and layer 2 datagram

service, using Nodes 1 and 2. Results are shown in Figures 3-7 and 3-8. Second, the effects of extraneous network traffic were measured using Node 1 sending to Node 2 while Node 3 transmitted spurious messages. Third, differences in delay characteristics among node pairs were measured using Node pairs 1 and 2, 2 and 3, and 3 and 1.

For all measurements at least 200 messages were sent, in order to provide reasonable sampling of delay times. Increasing the number of messages to 1000 did not significantly alter the observed distributions.

3.3. Layer 7 Results

Results for this set of experiments are summarized in the graphs of Figures 3-1 through 3-6, in which the vertical axes show file transfer rates in bytes per second, and the horizontal axes show file size (k = 1024, M = 1048576).

Figure 3-1 shows the transfer rates for a small-disk and a large-disk node.

Figure 3-2 shows transfer rates when two nodes are simultaneously transmitting to each other.

Figure 3-3 shows transfer rates when two nodes are simultaneously transmitting to a third node, which is otherwise unoccupied.

Figure 3-4 shows transfer rates when all three nodes are simultaneously transmitting to one node and receiving from another node.

Figure 3-5 shows transfer rates when two nodes are transmitting to a third node, which is simultaneously transmitting to one of the transmitting nodes.

Figure 3-6 shows the internal transfer rate, that is, the rate at which a file can be copied from one location to another in a node's memory. Note the change in vertical scale for this graph.

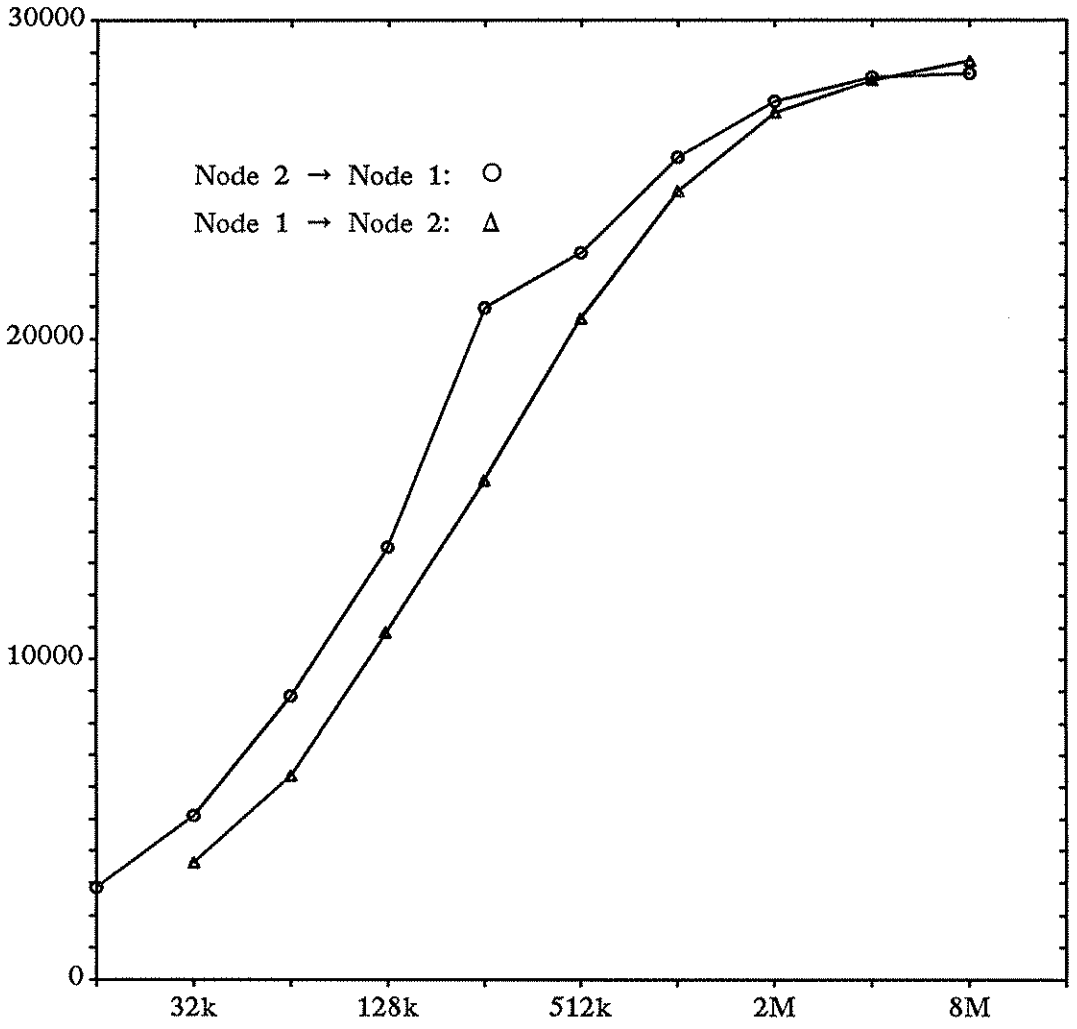


Figure 3-1.

Effect of Disk Size on Transfer Rate.

File Transfer Rate (bytes/sec) vs. File Size.

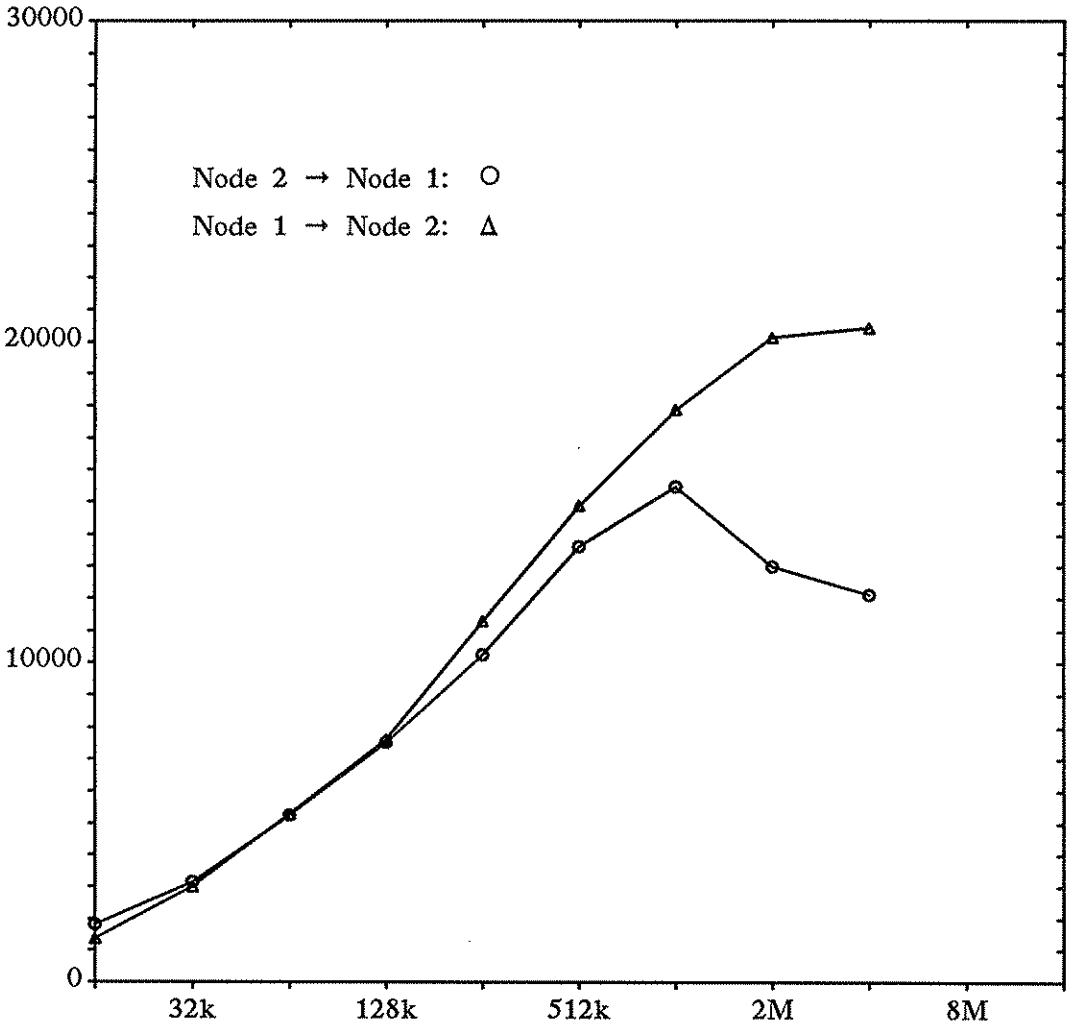


Figure 3-2.

Nodes 1 and 2 Transmitting Simultaneously.

File Transfer Rate (bytes/sec) vs. File Size.

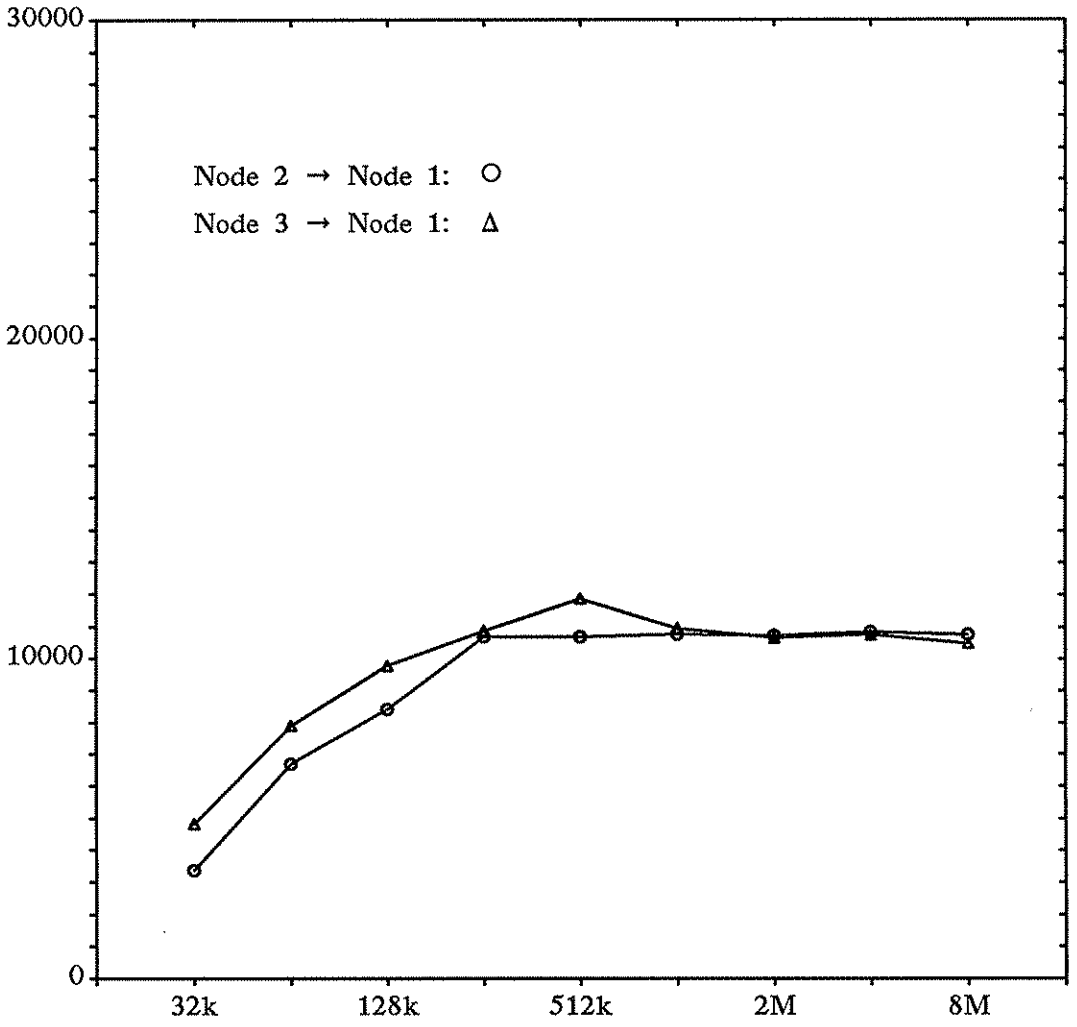


Figure 3-3.

Nodes 2 and 3 Transmitting Simultaneously to Node 1.

File Transfer Rate (bytes/sec) vs. File Size.

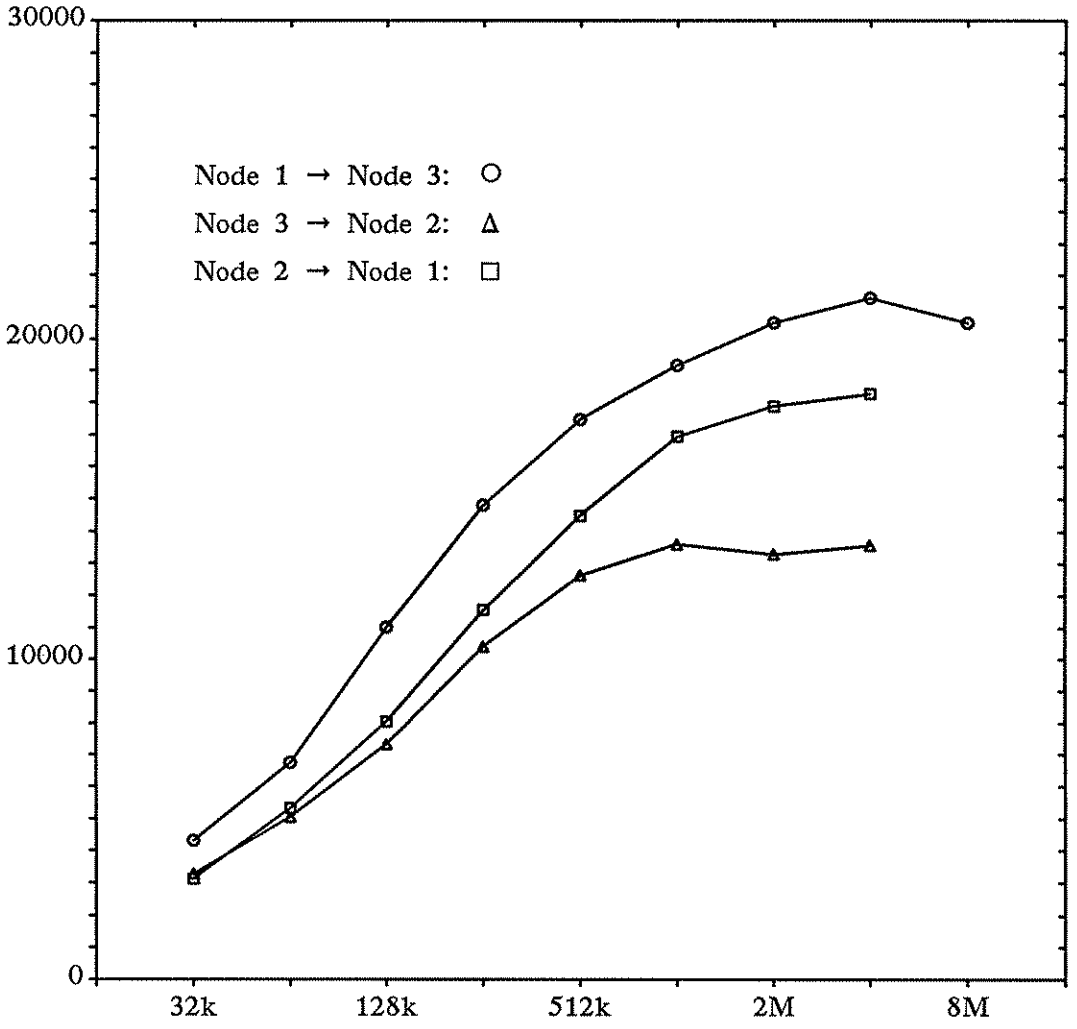


Figure 3-4.

Nodes 1, 2, and 3 Transmitting Simultaneously.

File Transfer Rate (bytes/sec) vs. File Size.

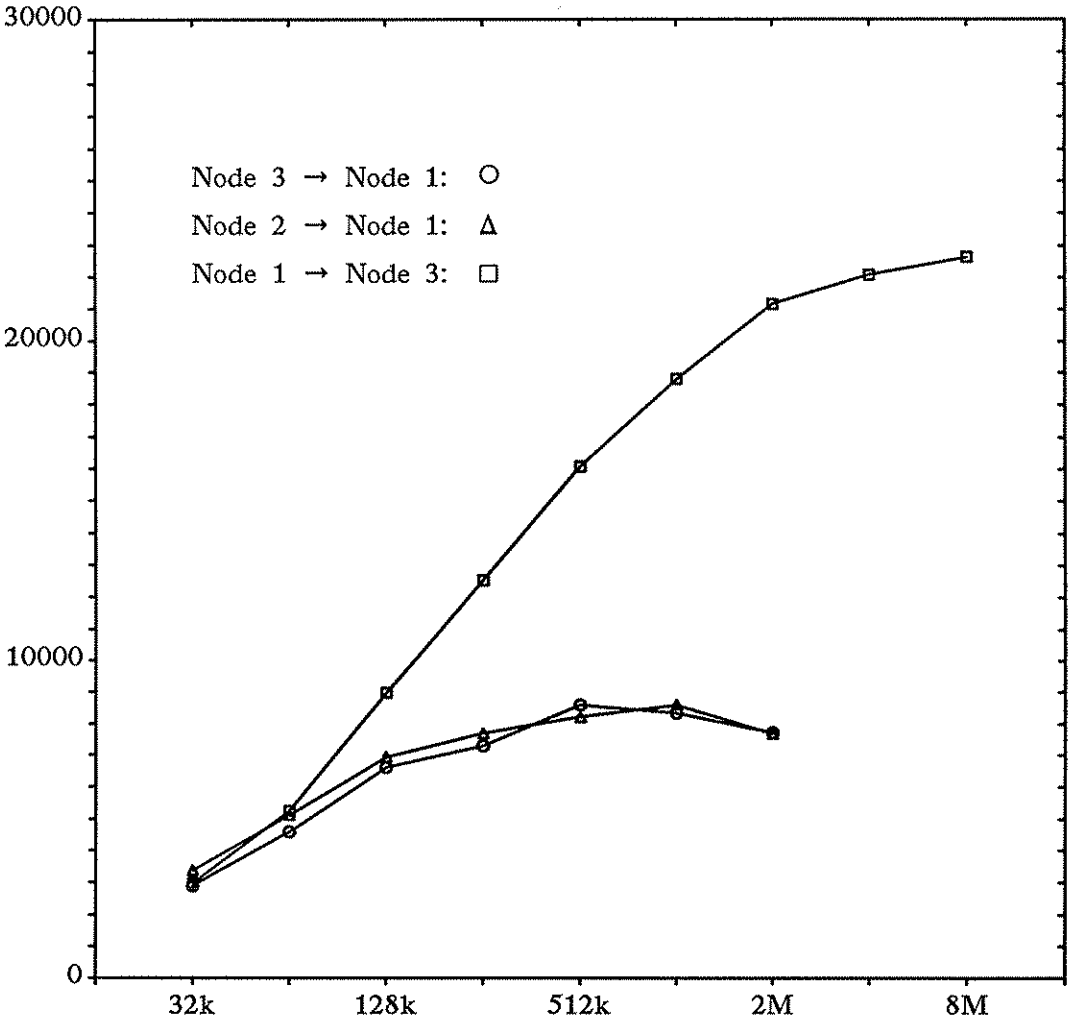


Figure 3-5.

Nodes 1 and 2 Transmitting to Node 1, Node 1 Transmitting to Node 3.

File Transfer Rate (bytes/sec) vs. File Size.

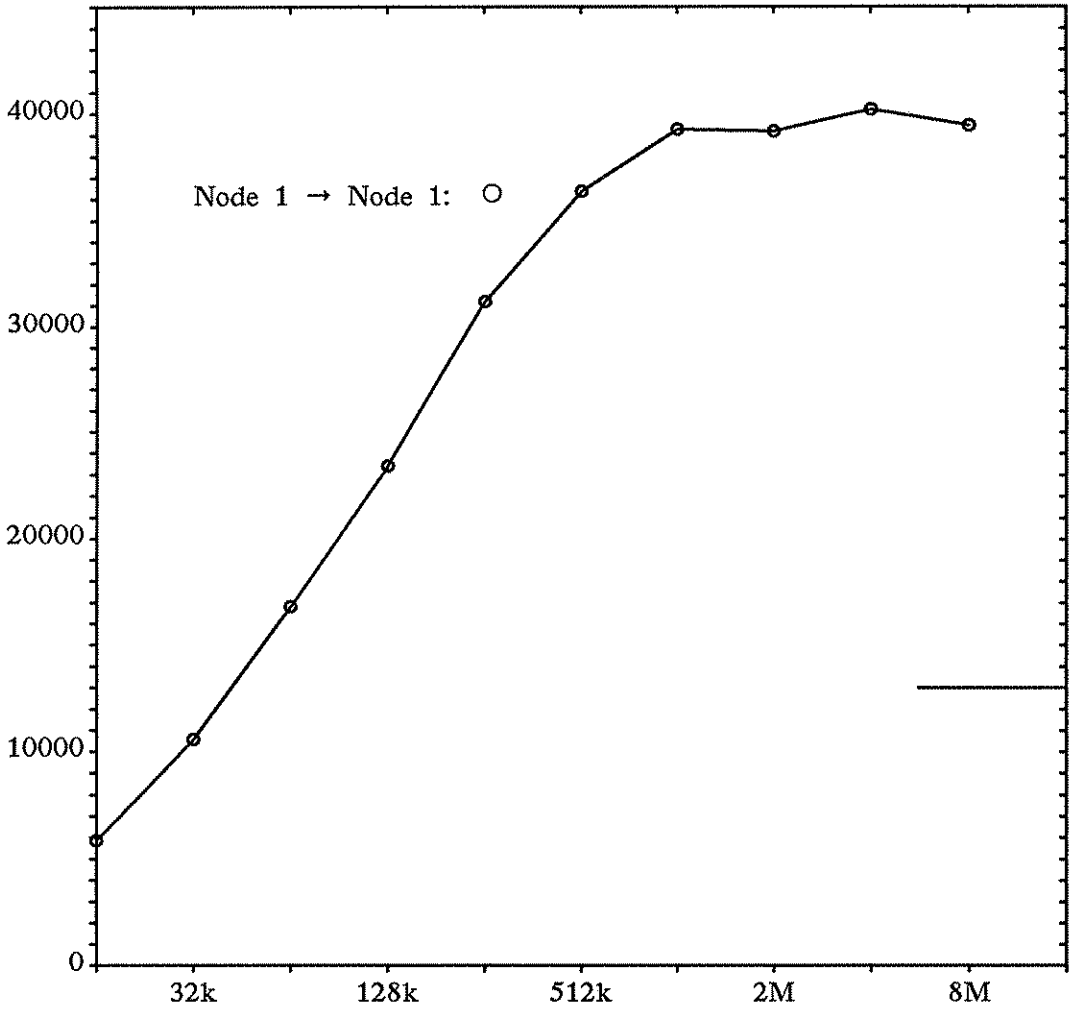


Figure 3-6.

Internal Transfer Rates for Node 1.

File Transfer Rate (bytes/sec) vs. File Size.

As the graphs show, transfer rates were much lower than the Ethernet capacity of 1,125,000 bytes per second. Total transfer rates never exceeded about 54,000 bytes per second, or 5% of capacity. Since collisions can be taken to be negligible at such low offered loads, the low channel utilization rate must be due to the internal file handling system and the characteristics of the iNA 961 software in conjunction with the iSBC 552 board.

It is clear that channel utilization would be much higher if the number of nodes were substantially increased. But 30,000 bytes per second appears to be a strong upper limit on the performance of an individual node, and even that can only be achieved when it is dedicated to a single activity.

The falloff observed in transfer rates for very large files can be attributed to the decreased efficiency of the file handler when disk space becomes scarce. It is interesting that the large-disk system suffered much more than the small-disk one, perhaps because the handler on the larger systems is more complex, and therefore slower.

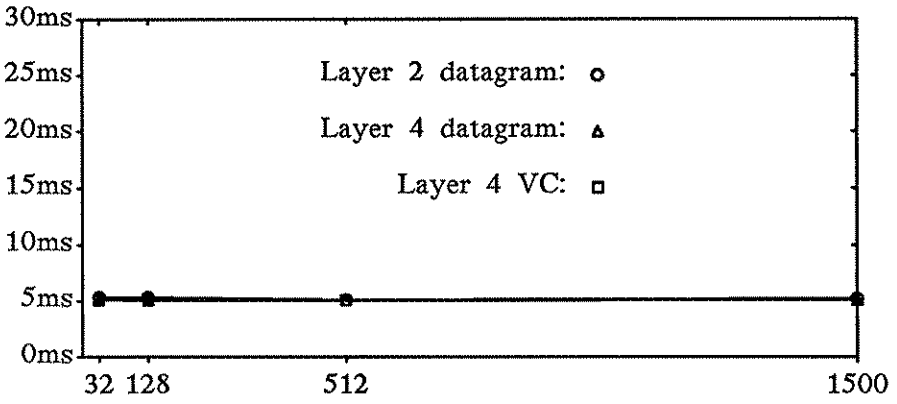
3.4. Layers 4 and 2 Results

Results for this layer are summarized in the following set of graphs, Figures 3-7 through 3-12.

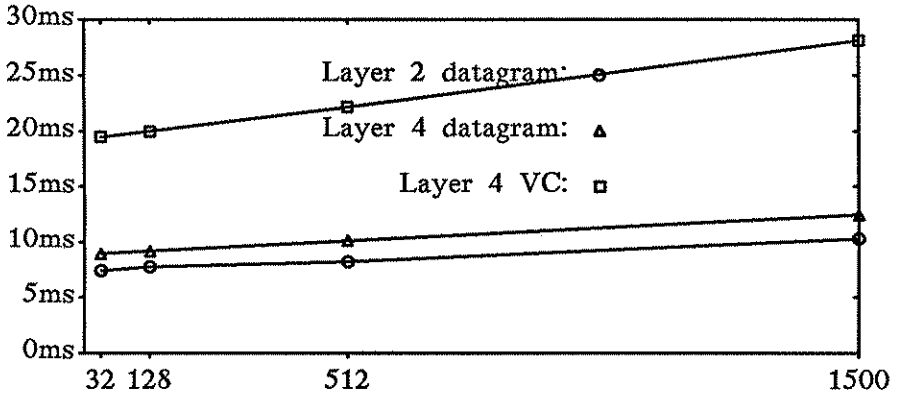
Figure 3-7 plots the mean times of request block delay, mailbox delay, and total node-to-node delay in milliseconds as a function of packet size, for layer 4 VC and datagram services, and layer 2 datagram service. Figures 3-8 (a) through (d) show the distributions of those means for 32 through 1500 bytes per packet, respectively. (All distribution graphs are discrete histograms, whose bar-widths are 0.1 millisecond. The bar-heights represent the observed frequencies relative to a maximum scale of 1.0.)

Figure 3-9 (a) plots layer 2 datagram delays with and without background traffic, and Figure 3-9 (b) plots layer 4 VC delays with and without background traffic. Figures 3-10 (a), (c), (e), and (g) show the distribution of the layer 2 means for packets of 32 through 1500 bytes; show the distribution of the VC means for the same packet sizes and traffic conditions.

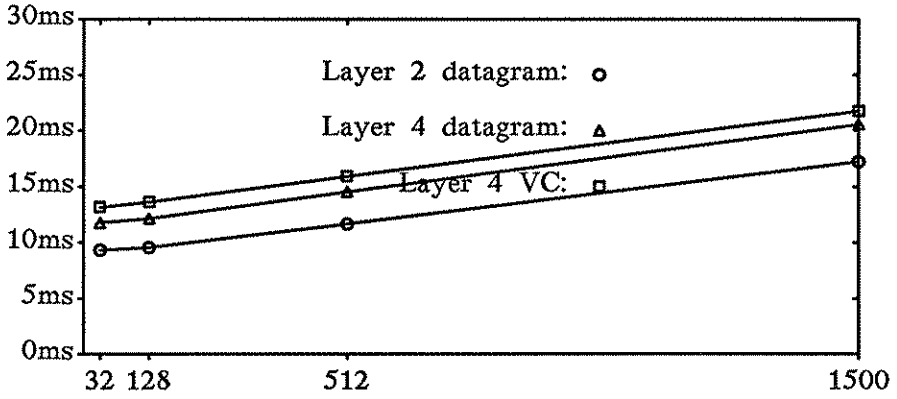
Figure 3-11 (a) plots layer 2 delays for all three combinations of sender-receiver node pairs, and Figure 3-11 (b) show the analogous plots for Layer 4 VC service. Figures 3-12 (a), (c), (e), and (g) show the distributions of the layer 2 means, and Figures 3-12 (b), (d), (f), and (h) show the distributions of the VC means, for the usual packet sizes of 32 through 1500 bytes.



CqCommRB Request Block Delay



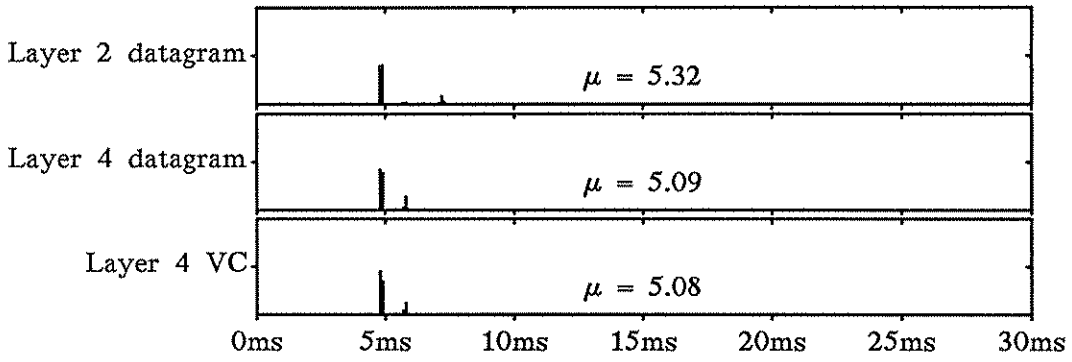
RqReceiveMessage Response Delay



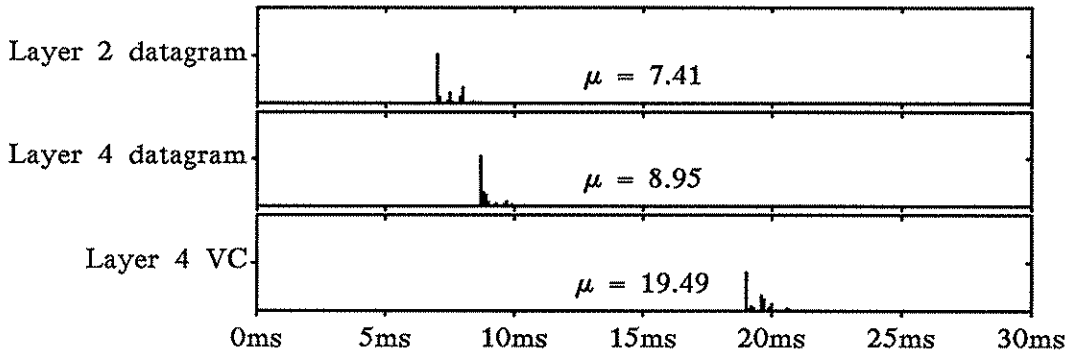
End to End Delay

Figure 3-7.

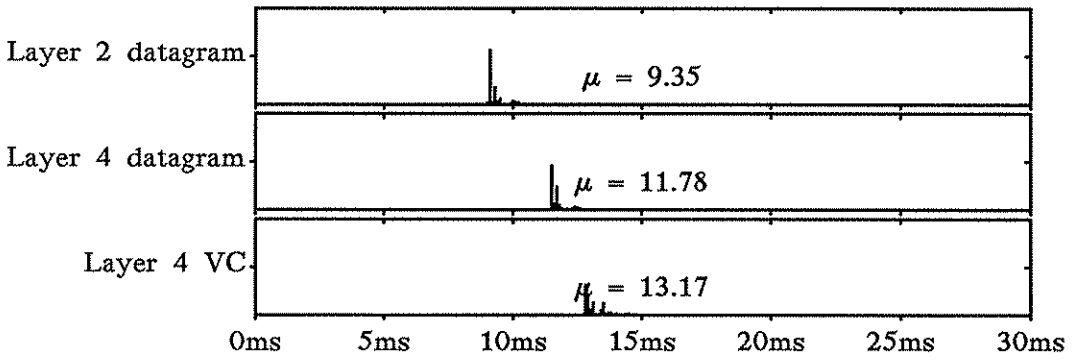
Effect of Protocol Layer on Mean Delays.



CqCommRB Request Block Delay



RqReceiveMessage Response Delay

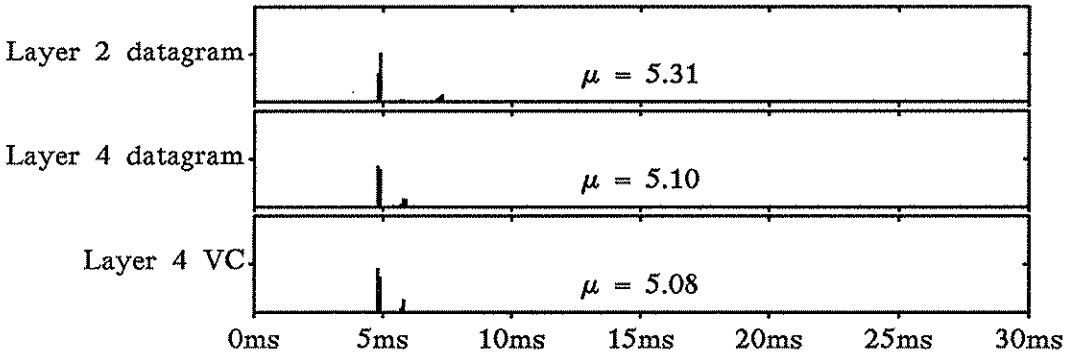


End to End Delay

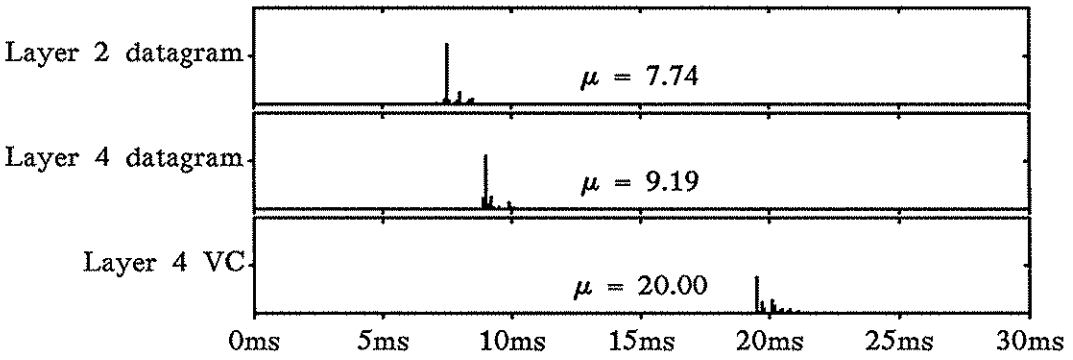
Figure 3-8 (a).

Effect of Protocol Layer on Delay Distributions.

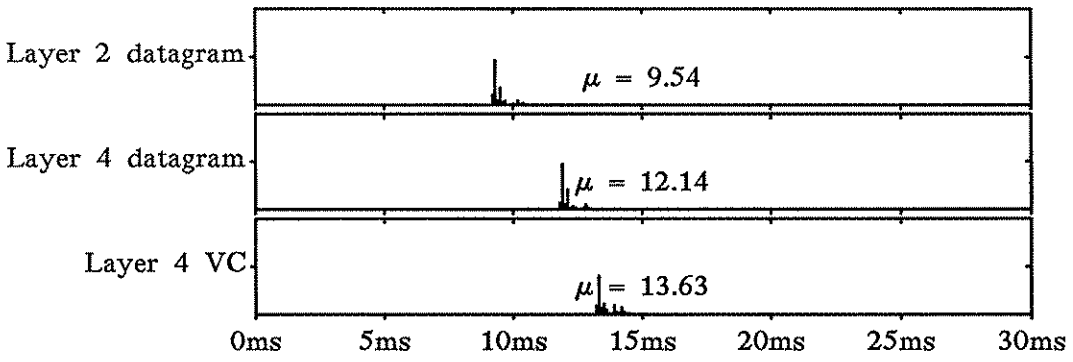
(32 Bytes/Packet)



CqCommRB Request Block Delay



RqReceiveMessage Response Delay

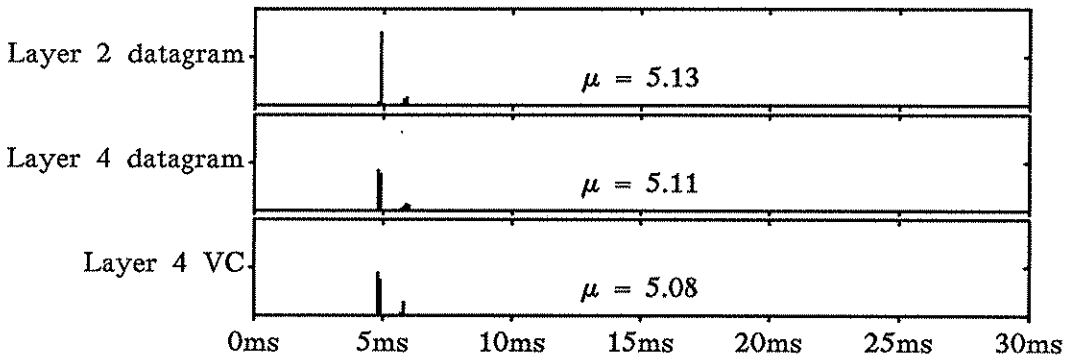


End to End Delay

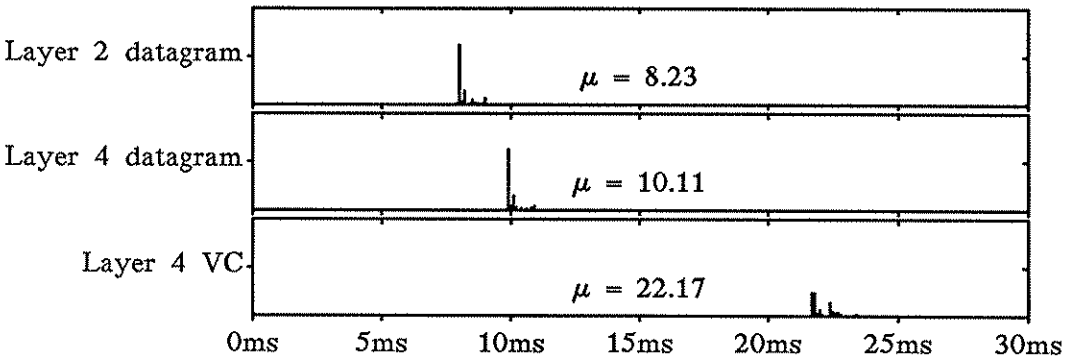
Figure 3-8 (b).

Effect of Protocol Layer on Delay Distributions.

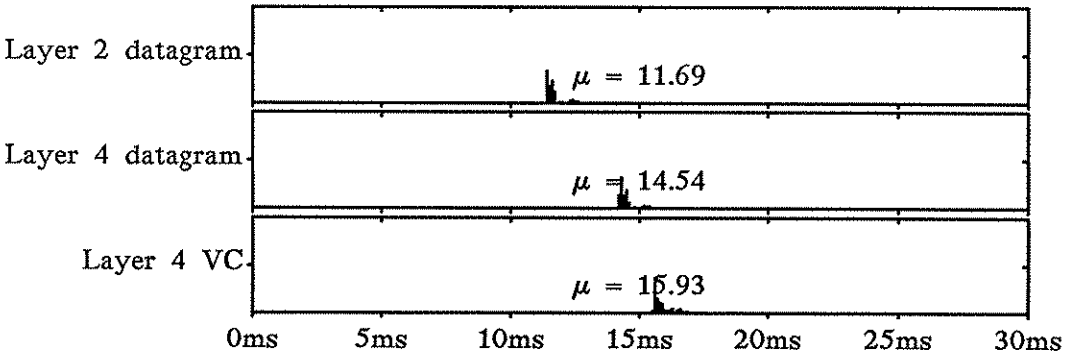
(128 Bytes/Packet)



CqCommRB Request Block Delay



RqReceiveMessage Response Delay

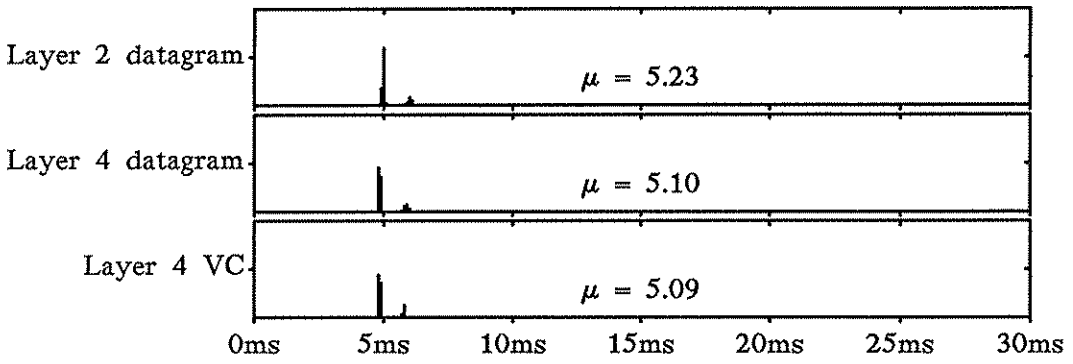


End to End Delay

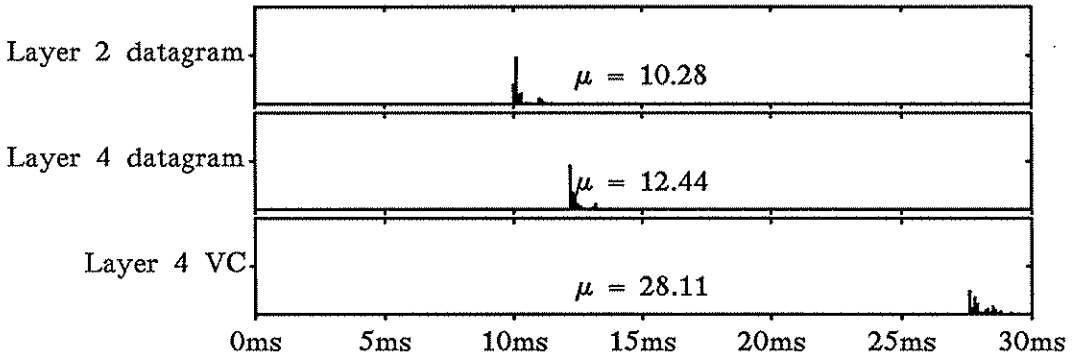
Figure 3-8 (c).

Effect of Protocol Layer on Delay Distributions.

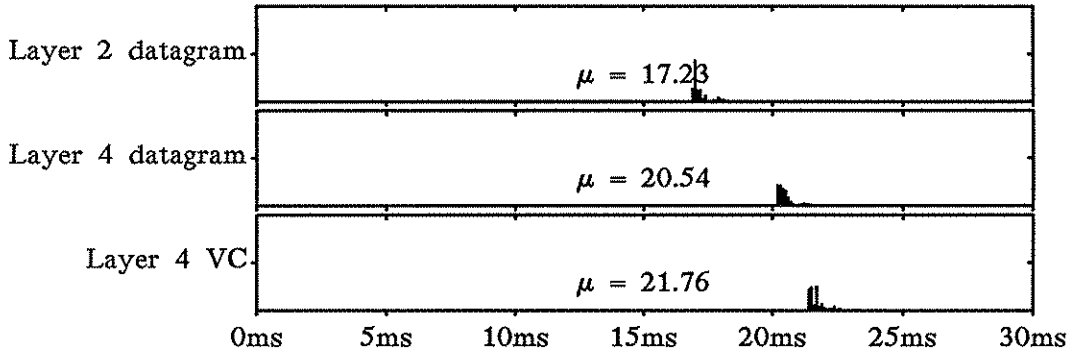
(512 Bytes/Packet)



CqCommRB Request Block Delay



RqReceiveMessage Response Delay

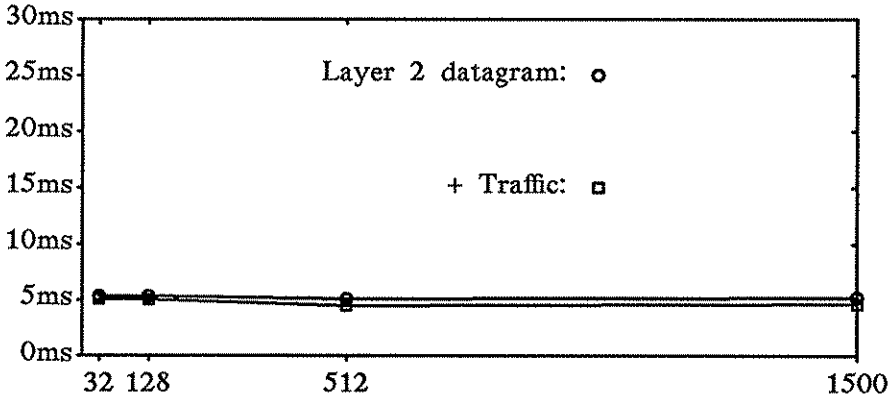


End to End Delay

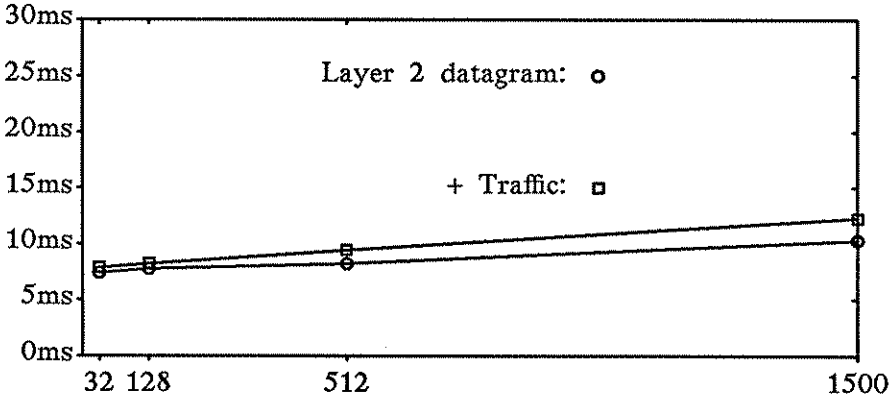
Figure 3-8 (d).

Effect of Protocol Layer on Delay Distributions.

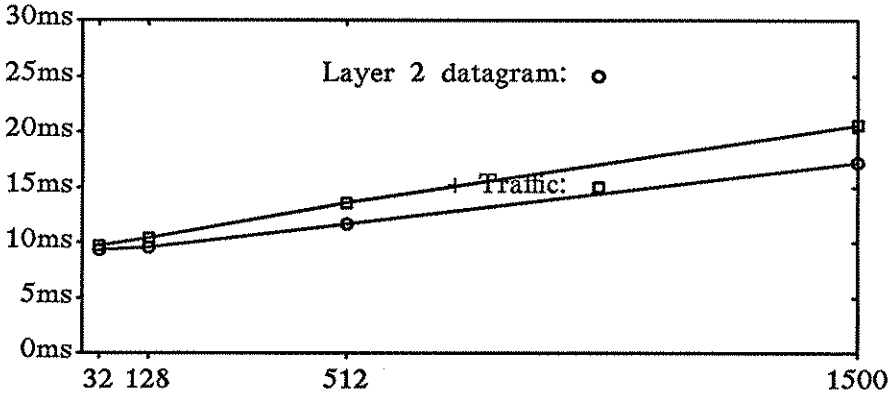
(1500 Bytes/Packet)



CqCommRB Request Block Delay



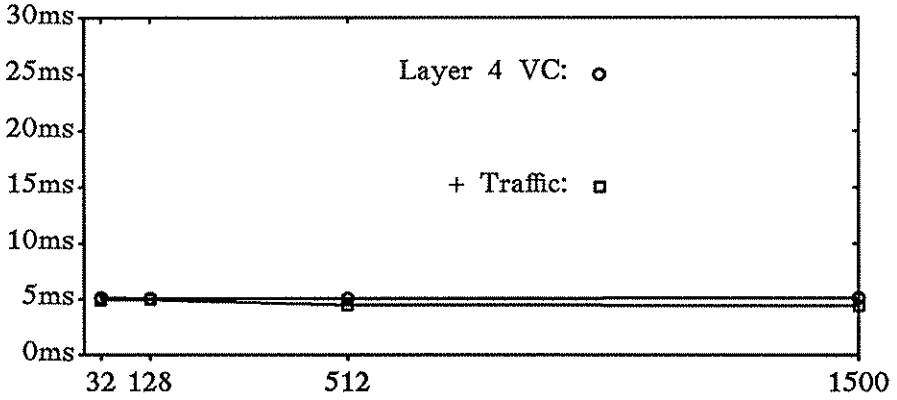
RqReceiveMessage Response Delay



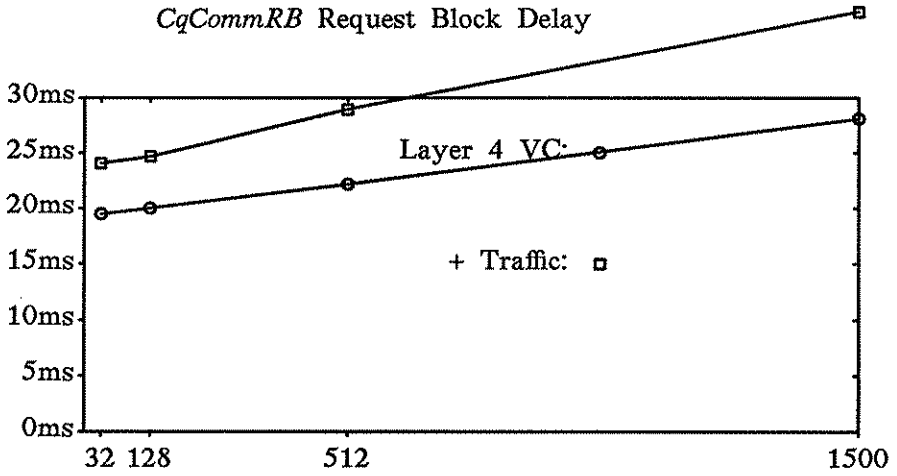
End to End Delay

Figure 3-9 (a).

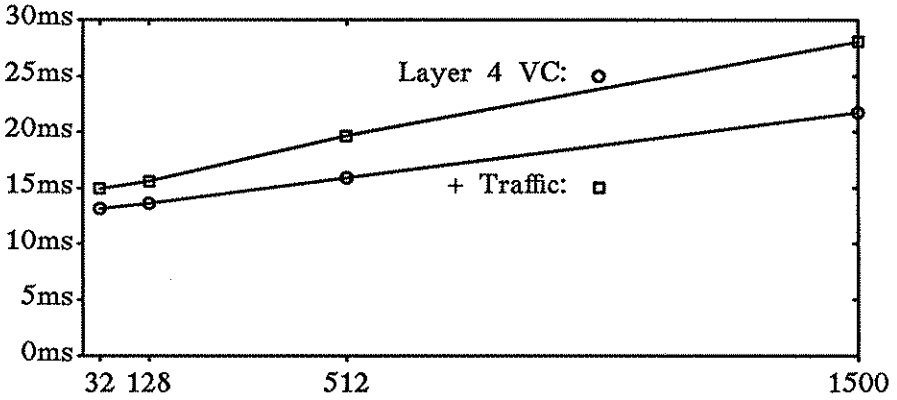
Effect of Traffic on Mean Delays.



CqCommRB Request Block Delay



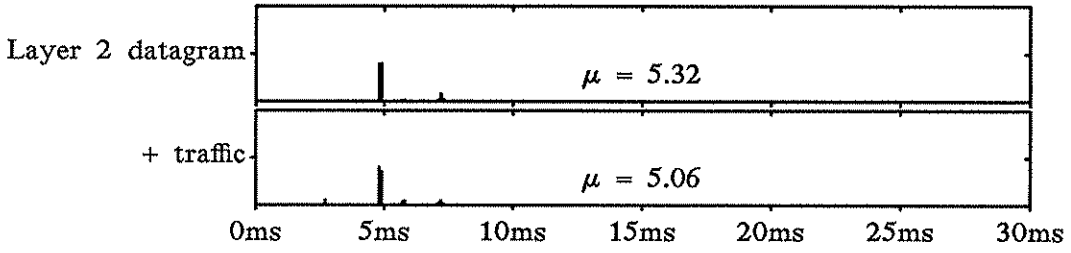
RqReceiveMessage Response Delay



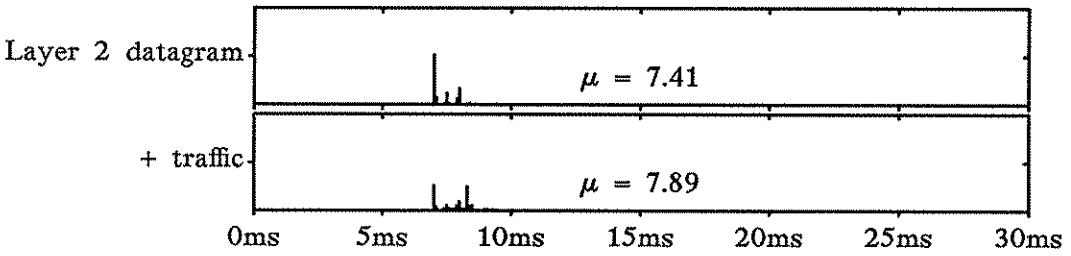
End to End Delay

Figure 3-9 (b).

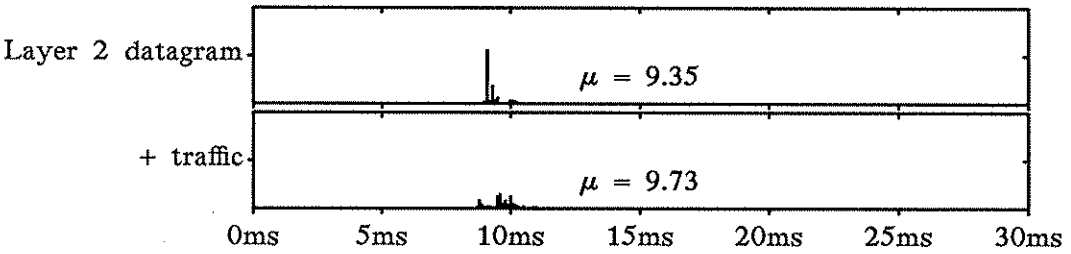
Effect of Traffic on Mean Delays.



CqCommRB Request Block Delay



RqReceiveMessage Response Delay

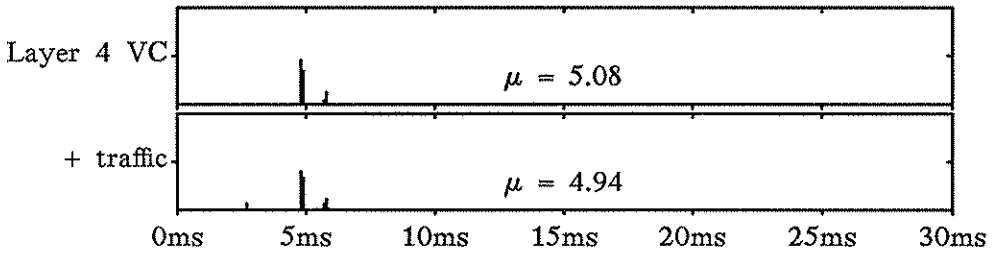


End to End Delay

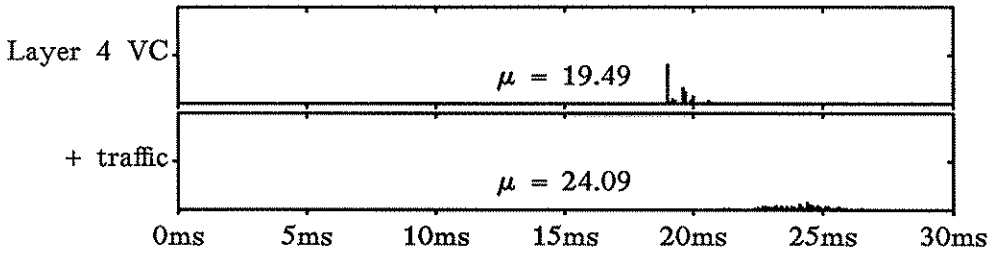
Figure 3-10 (a).

Effect of Traffic on Delay Distributions.

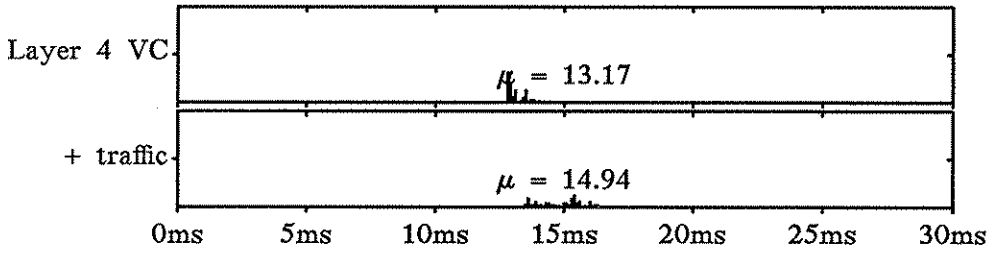
(32 Bytes/Packet)



CqCommRB Request Block Delay



RqReceiveMessage Response Delay

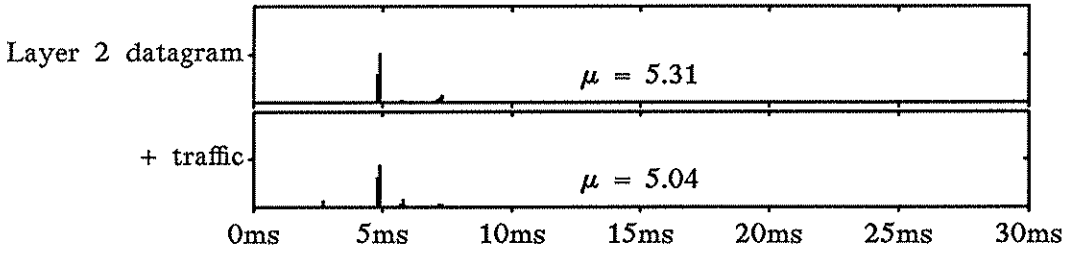


End to End Delay

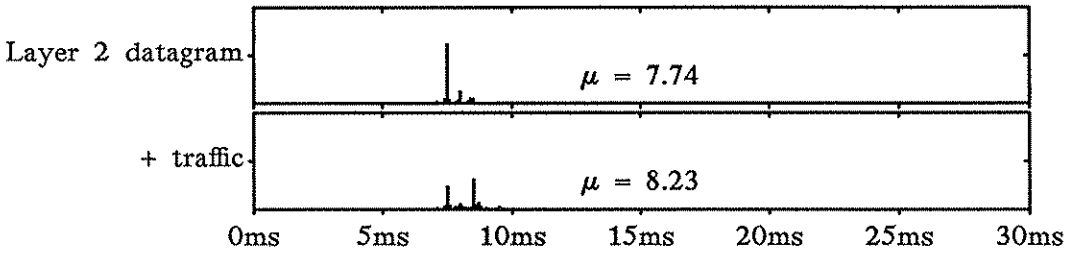
Figure 3-10 (b).

Effect of Traffic on Delay Distributions.

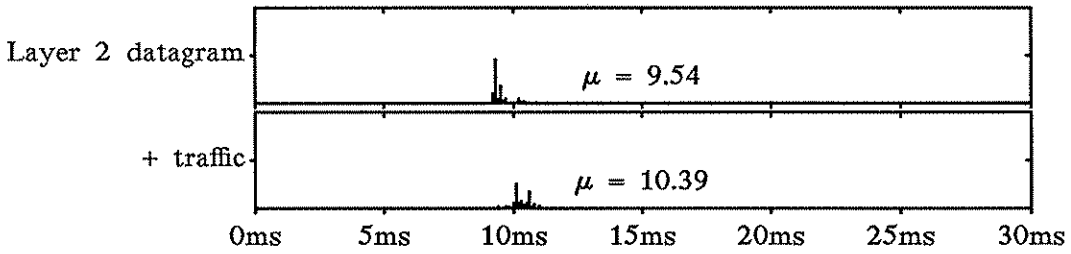
(32 Bytes/Packet)



CqCommRB Request Block Delay



RqReceiveMessage Response Delay

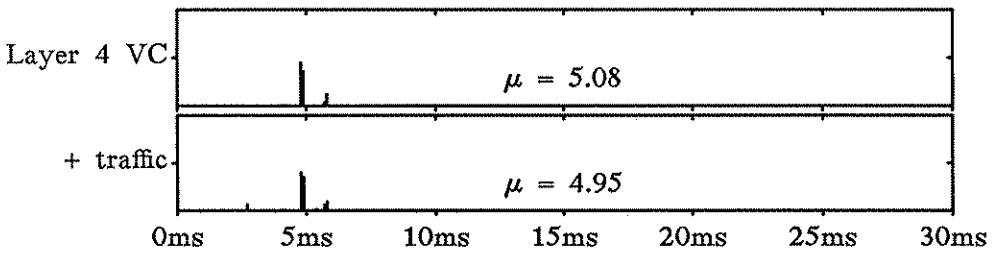


End to End Delay

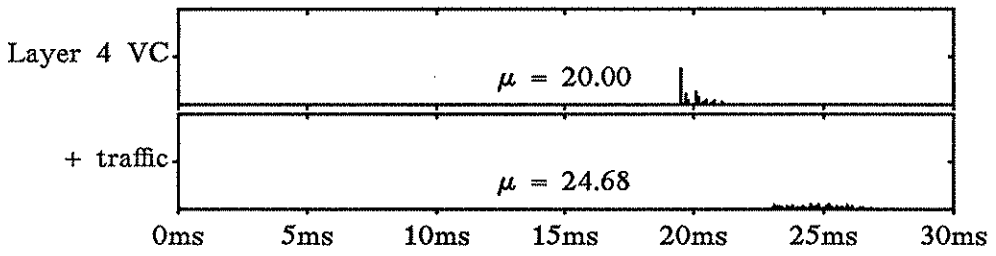
Figure 3-10 (c).

Effect of Traffic on Delay Distributions.

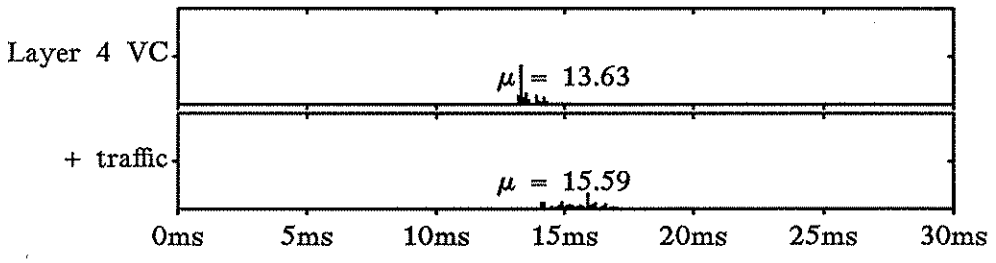
(128 Bytes/Packet)



CqCommRB Request Block Delay



RqReceiveMessage Response Delay

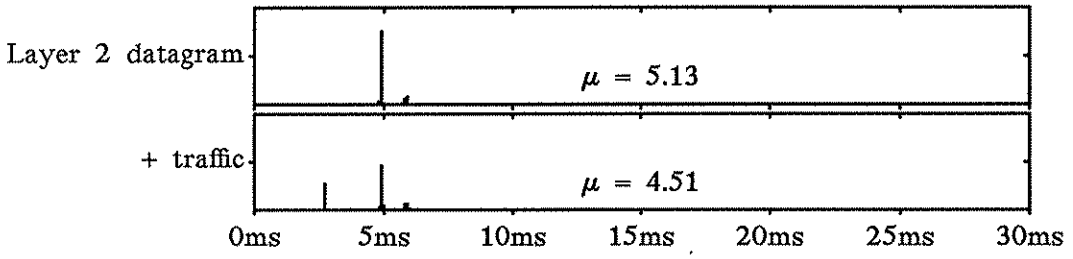


End to End Delay

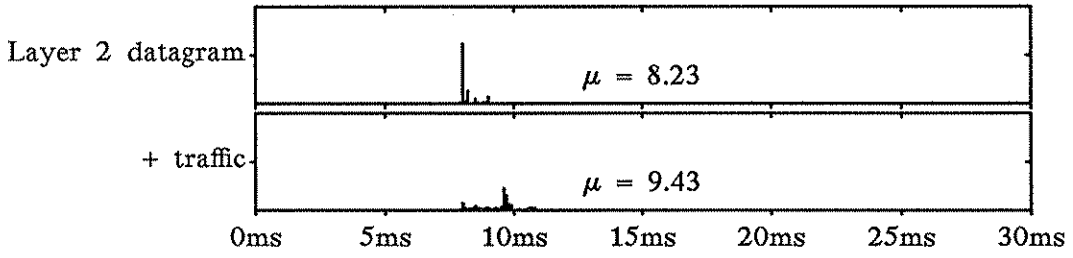
Figure 3-10 (d).

Effect of Traffic on Delay Distributions.

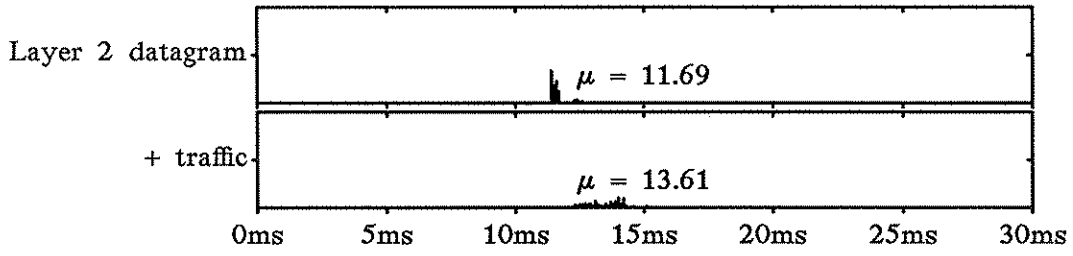
(128 Bytes/Packet)



CqCommRB Request Block Delay



RqReceiveMessage Response Delay

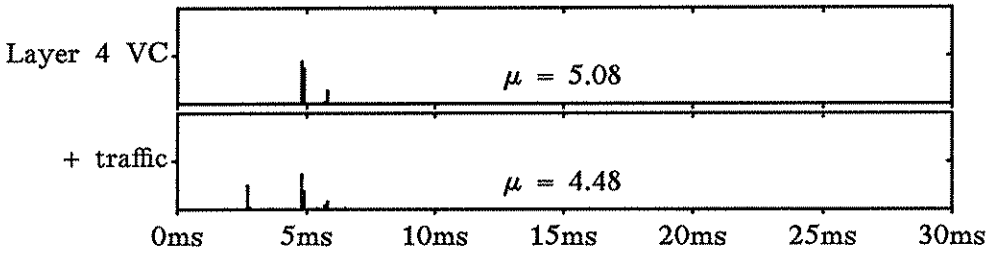


End to End Delay

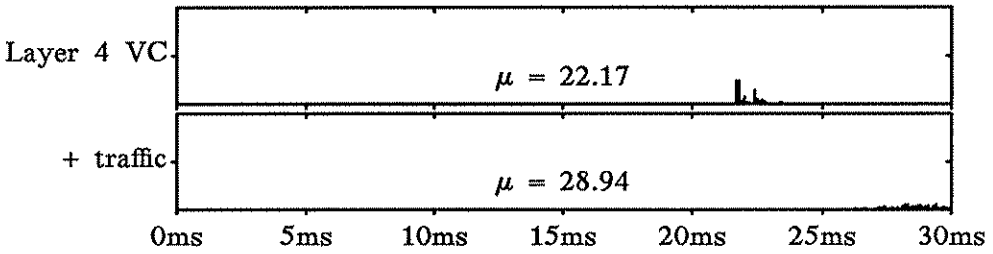
Figure 3-10 (e).

Effect of Traffic on Delay Distributions.

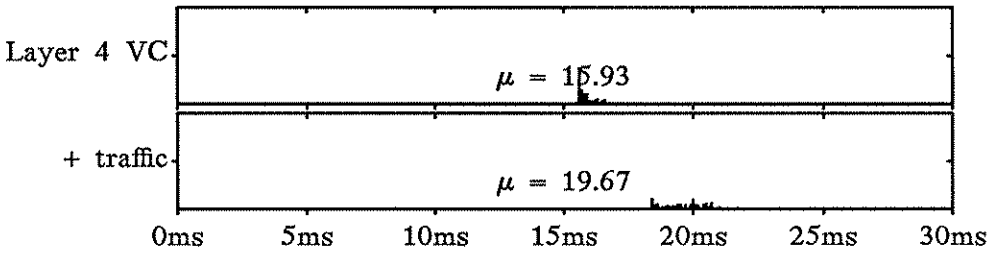
(512 Bytes/Packet)



CqCommRB Request Block Delay



RqReceiveMessage Response Delay

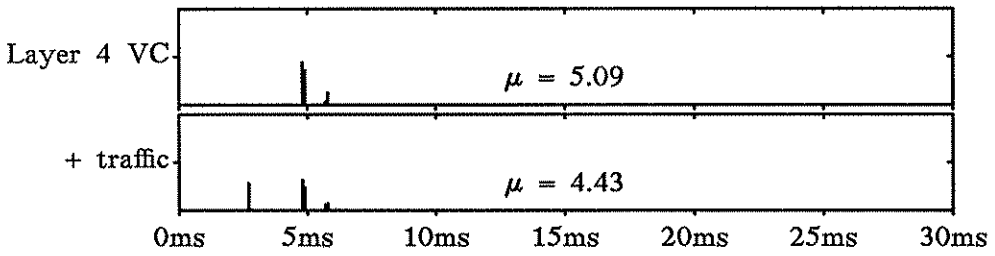


End to End Delay

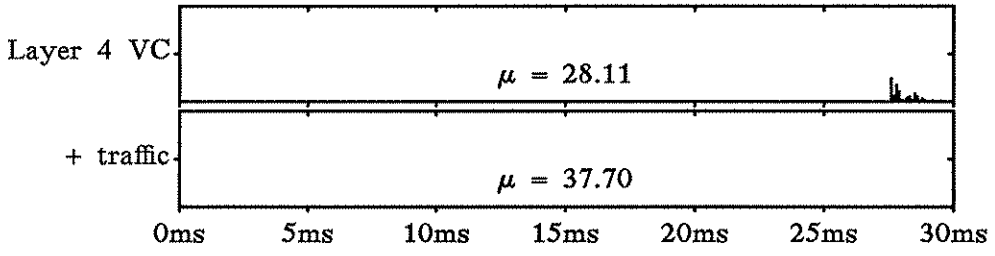
Figure 3-10 (f).

Effect of Traffic on Delay Distributions.

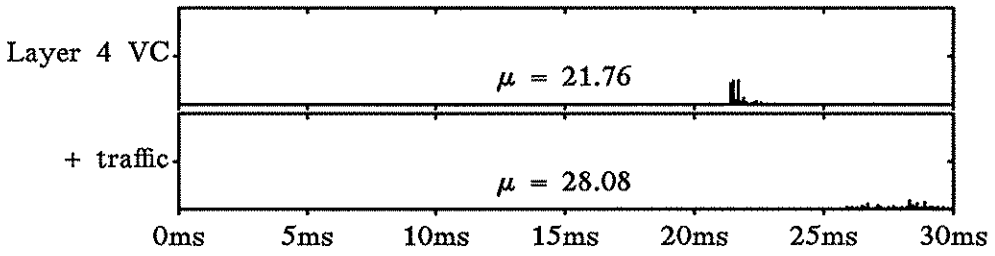
(512 Bytes/Packet)



CqCommRB Request Block Delay



RqReceiveMessage Response Delay

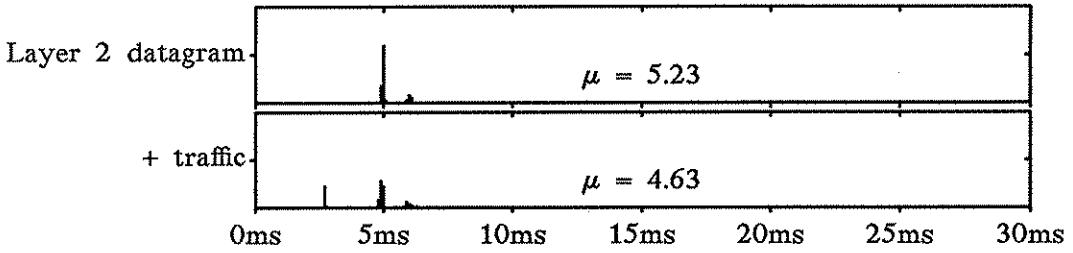


End to End Delay

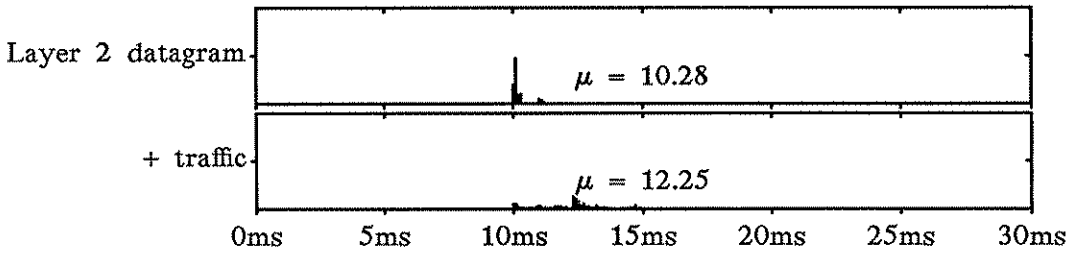
Figure 3-10 (h).

Effect of Traffic on Delay Distributions.

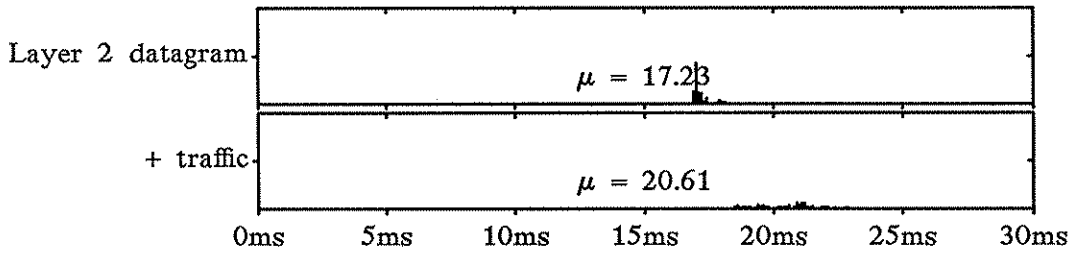
(1500 Bytes/Packet)



CqCommRB Request Block Delay



RqReceiveMessage Response Delay

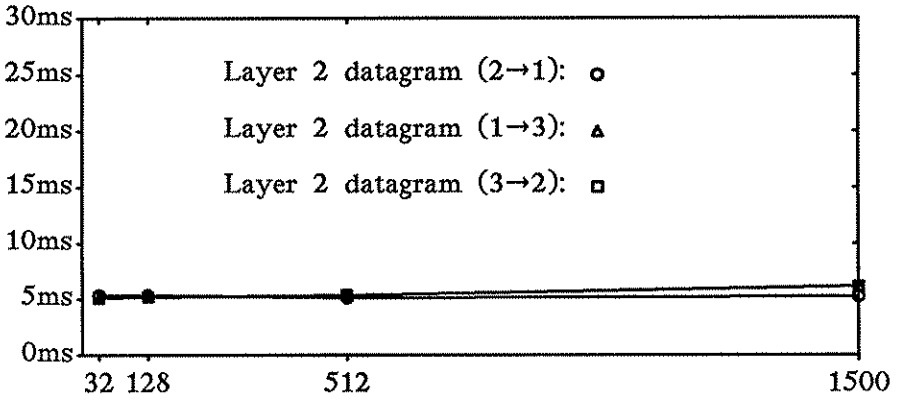


End to End Delay

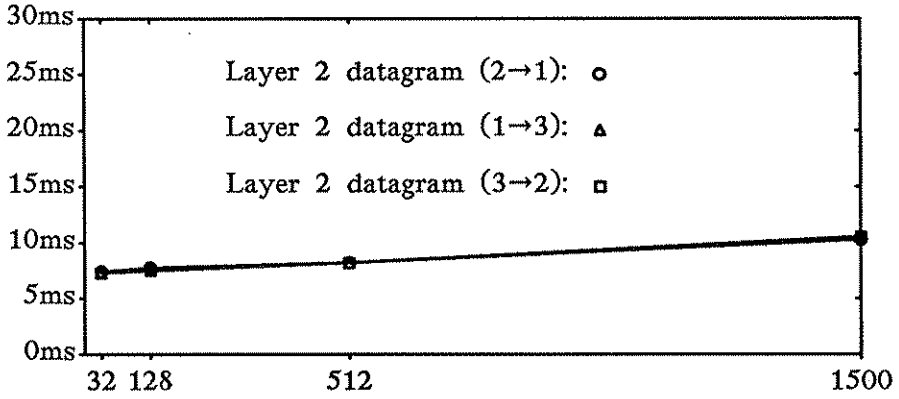
Figure 3-10 (g).

Effect of Traffic on Delay Distributions.

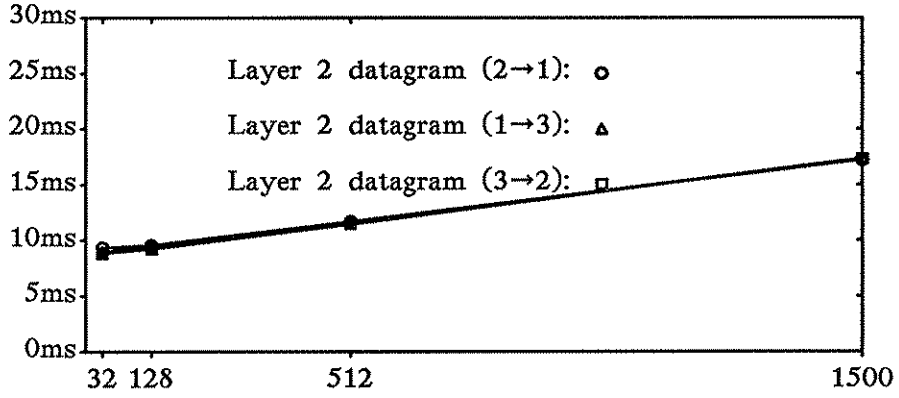
(1500 Bytes/Packet)



CqCommRB Request Block Delay



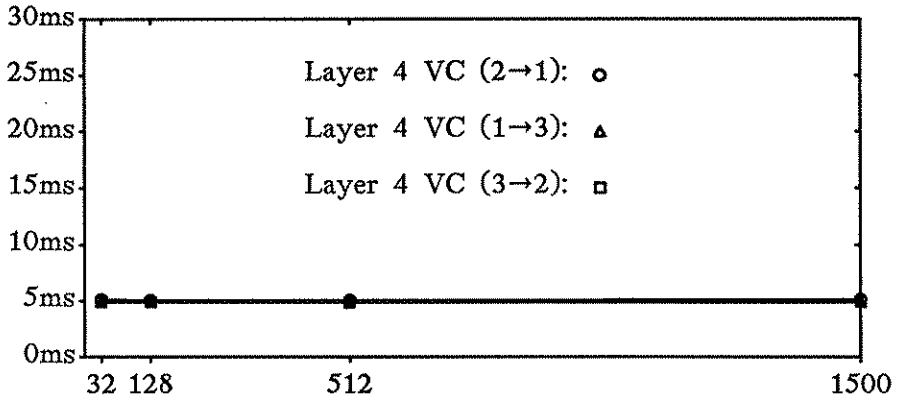
RqReceiveMessage Response Delay



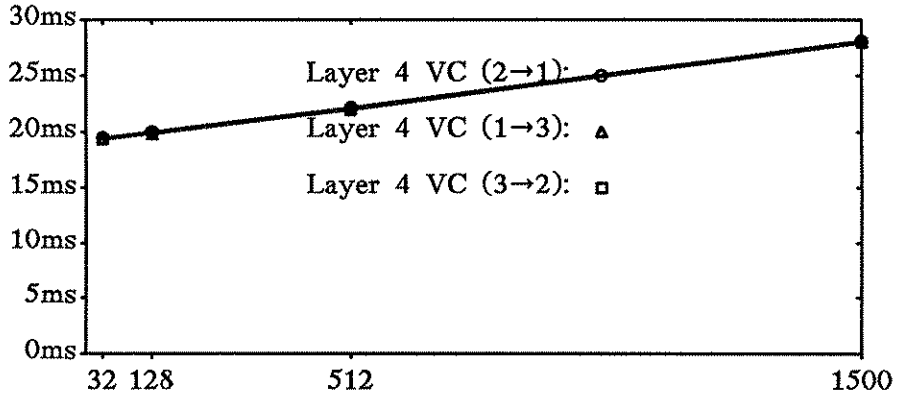
End to End Delay

Figure 3-11 (a).

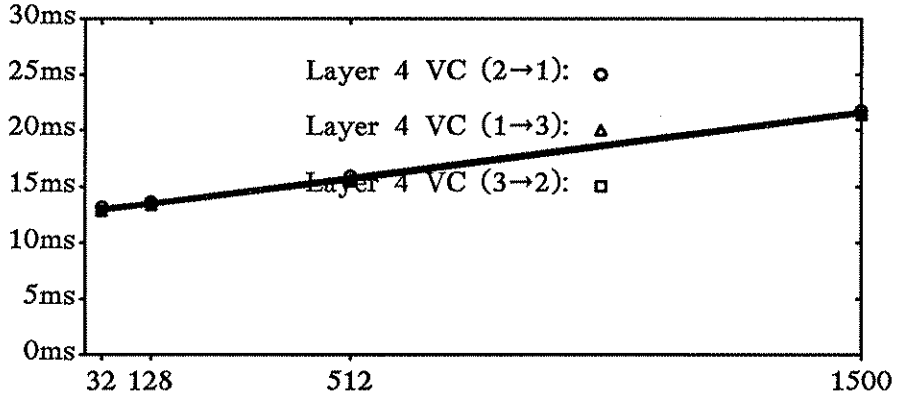
Effect of Configuration on Mean Delays.



CqCommRB Request Block Delay



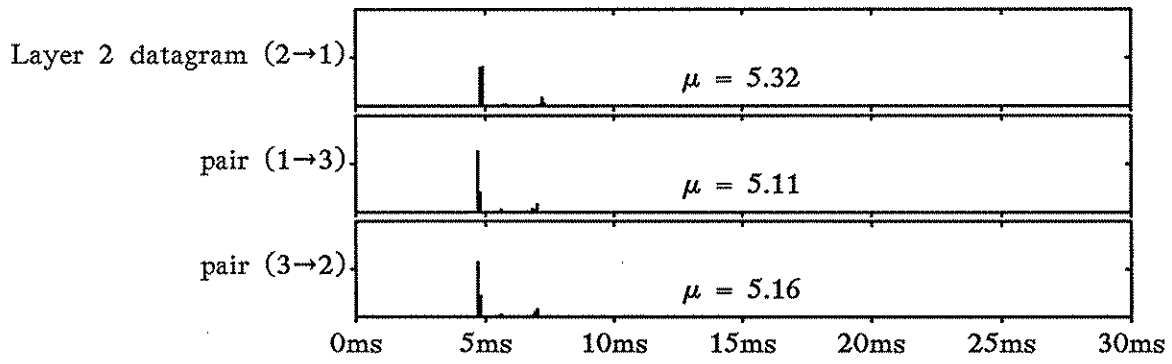
RqReceiveMessage Response Delay



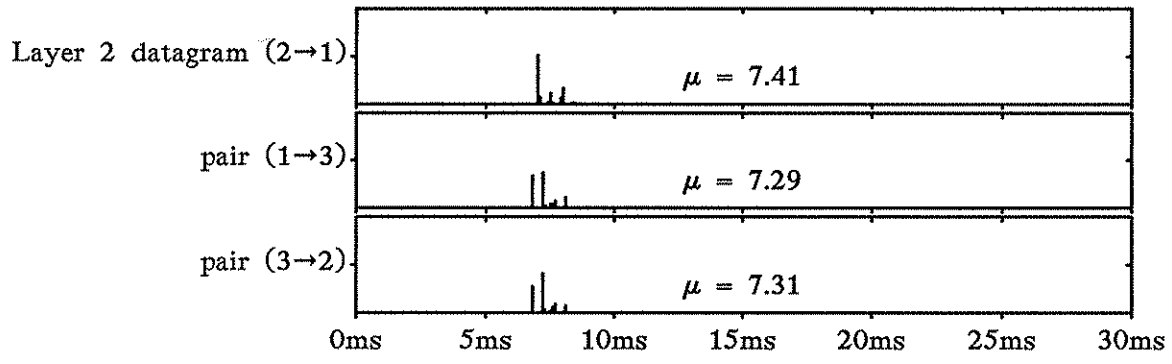
End to End Delay

Figure 3-11 (b).

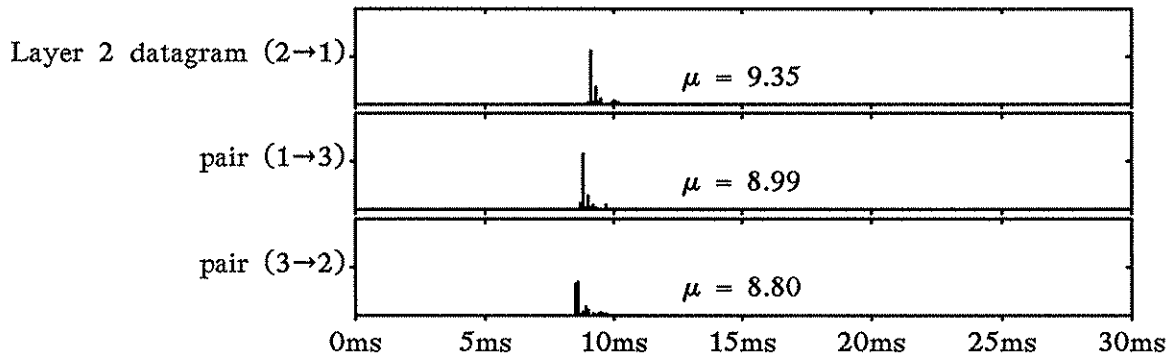
Effect of Configuration on Mean Delays.



CqCommRB Request Block Delay



RqReceiveMessage Response Delay

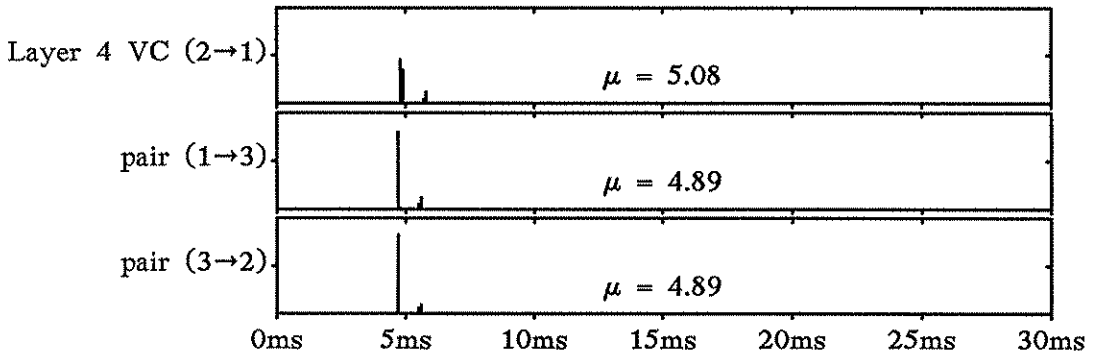


End to End Delay

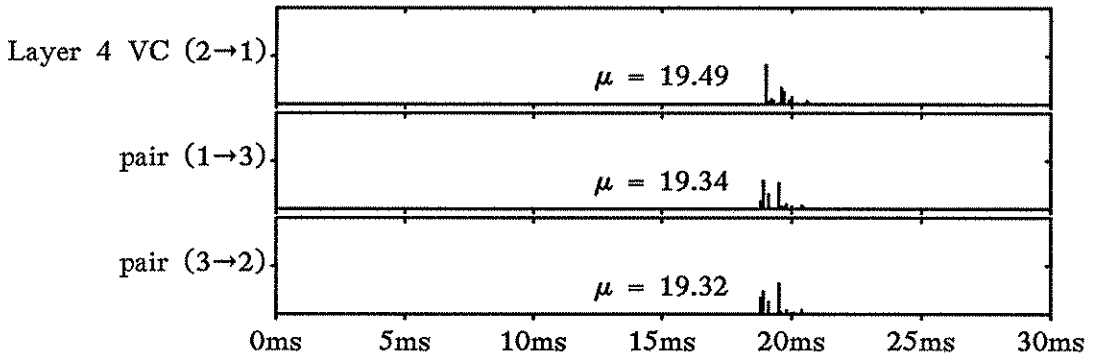
Figure 3-12 (a).

Effect of Configuration on Delay Distributions.

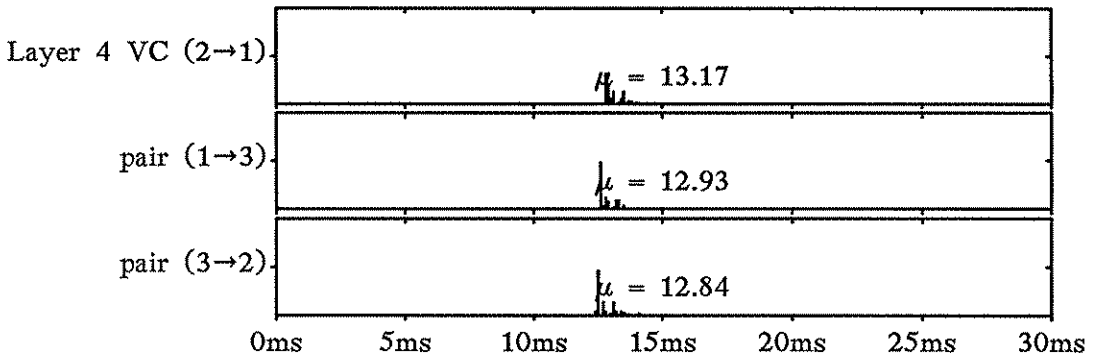
(32 Bytes/Packet)



CqCommRB Request Block Delay



RqReceiveMessage Response Delay

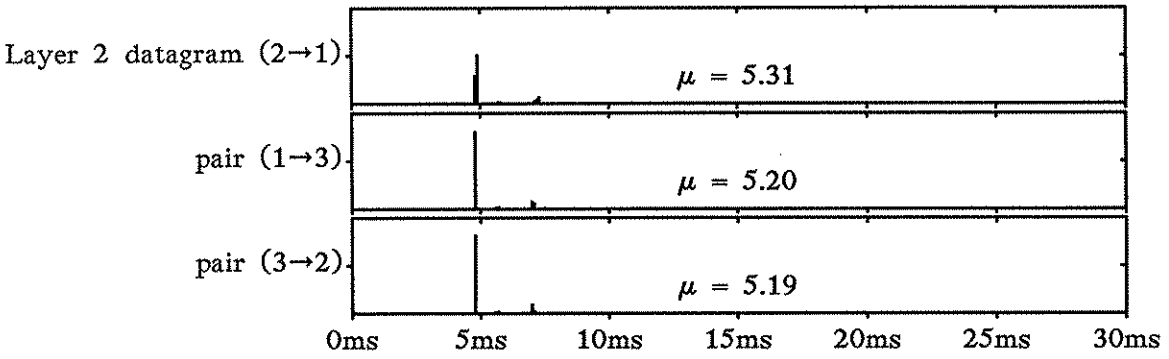


End to End Delay

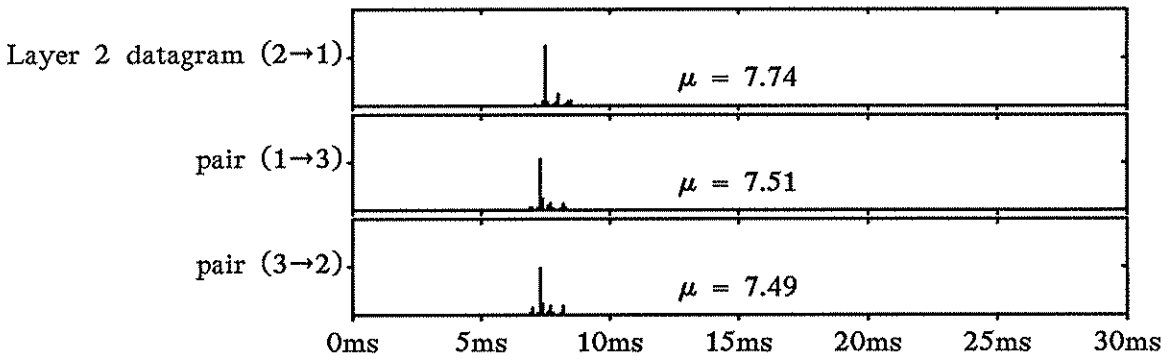
Figure 3-12 (b).

Effect of Configuration on Delay Distributions.

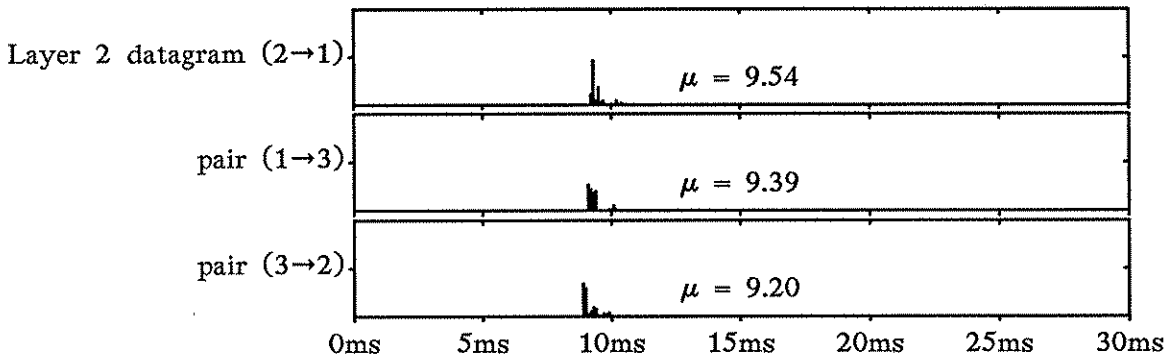
(32 Bytes/Packet)



CqCommRB Request Block Delay



RqReceiveMessage Response Delay

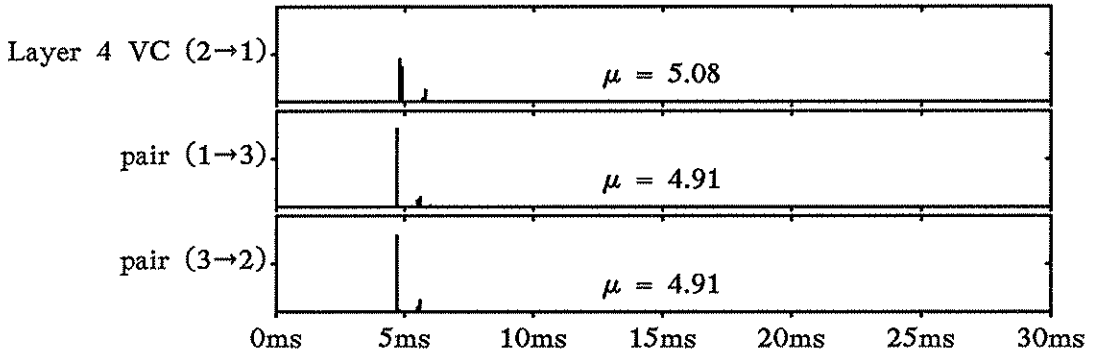


End to End Delay

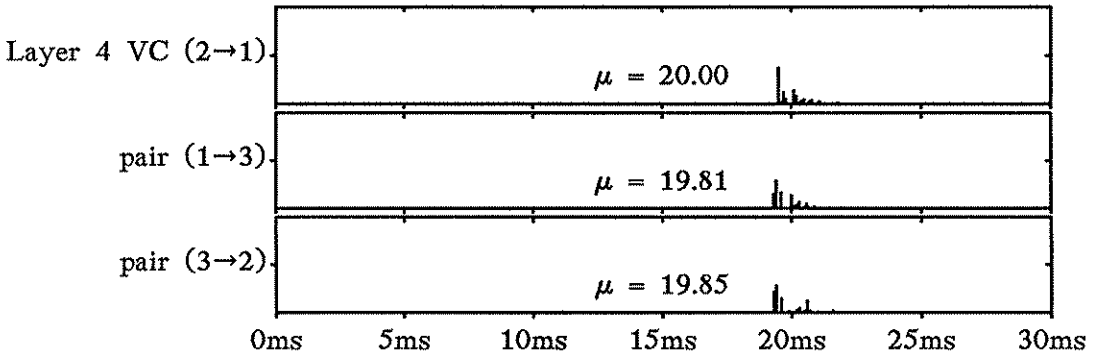
Figure 3-12 (c).

Effect of Configuration on Delay Distributions.

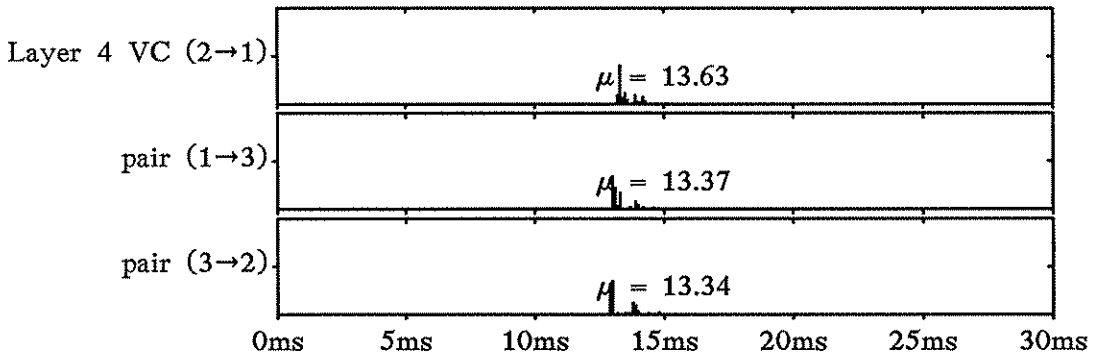
(128 Bytes/Packet)



CqCommRB Request Block Delay



RqReceiveMessage Response Delay

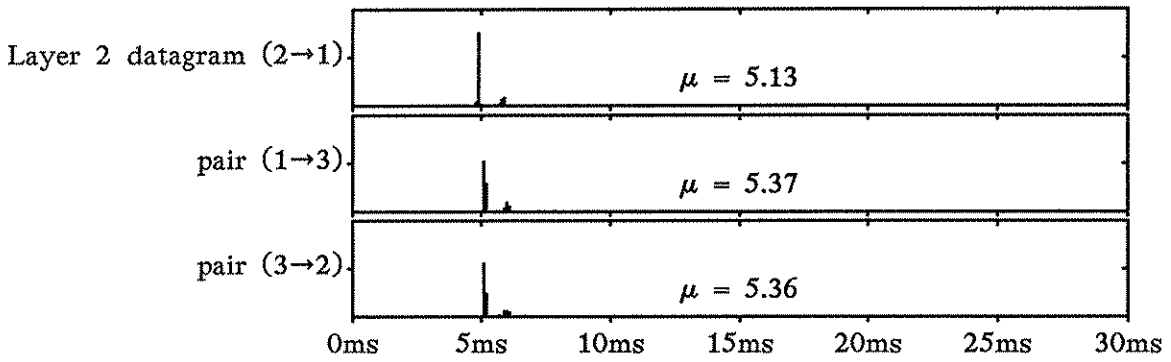


End to End Delay

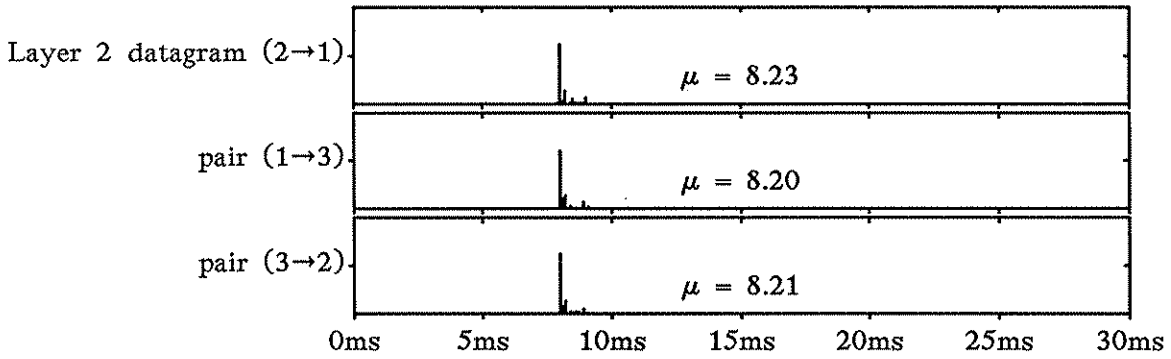
Figure 3-12 (d).

Effect of Configuration on Delay Distributions.

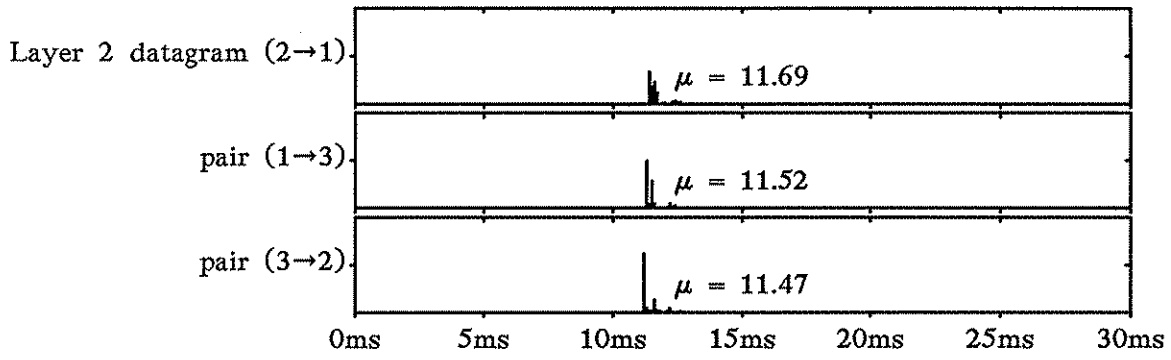
(128 Bytes/Packet)



CqCommRB Request Block Delay



RqReceiveMessage Response Delay

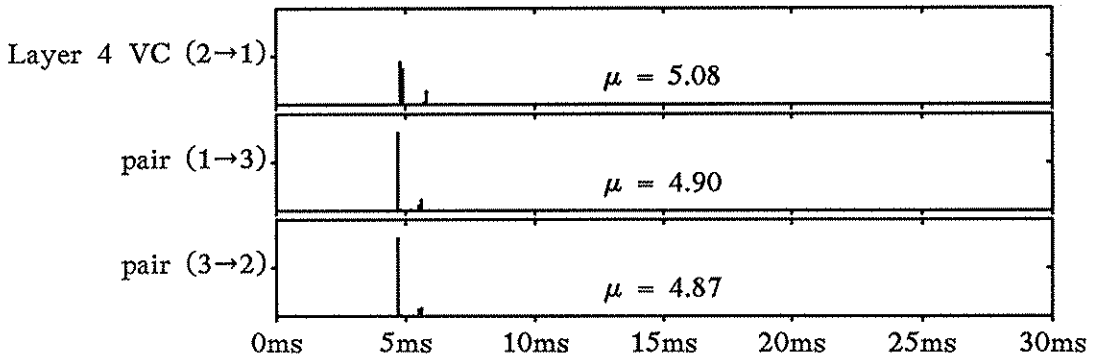


End to End Delay

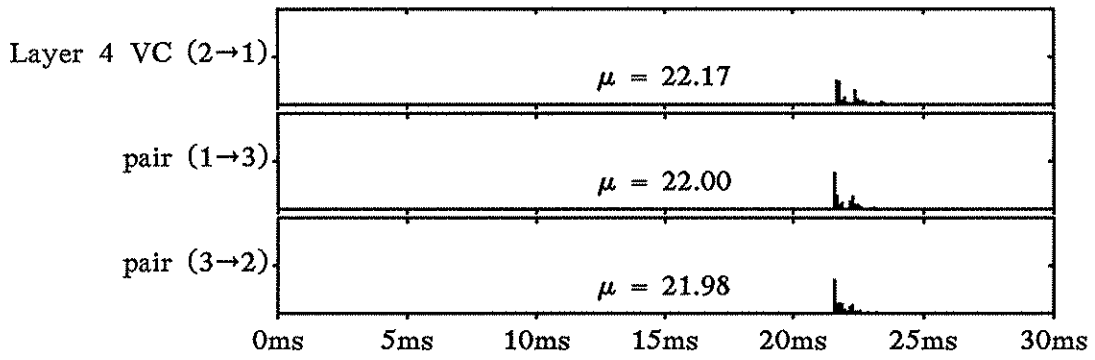
Figure 3-12 (e).

Effect of Configuration on Delay Distributions.

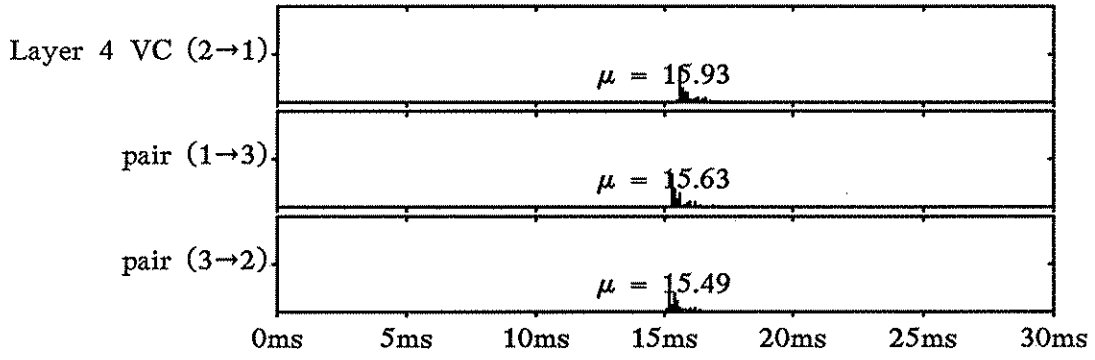
(512 Bytes/Packet)



CqCommRB Request Block Delay



RqReceiveMessage Response Delay

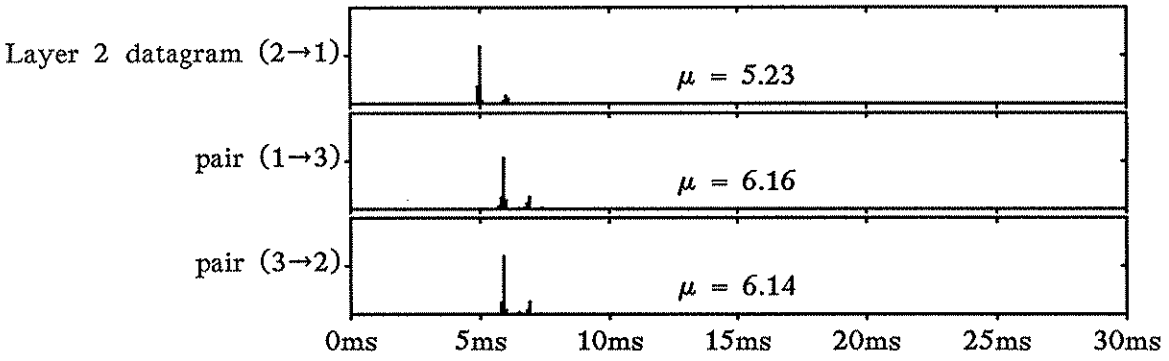


End to End Delay

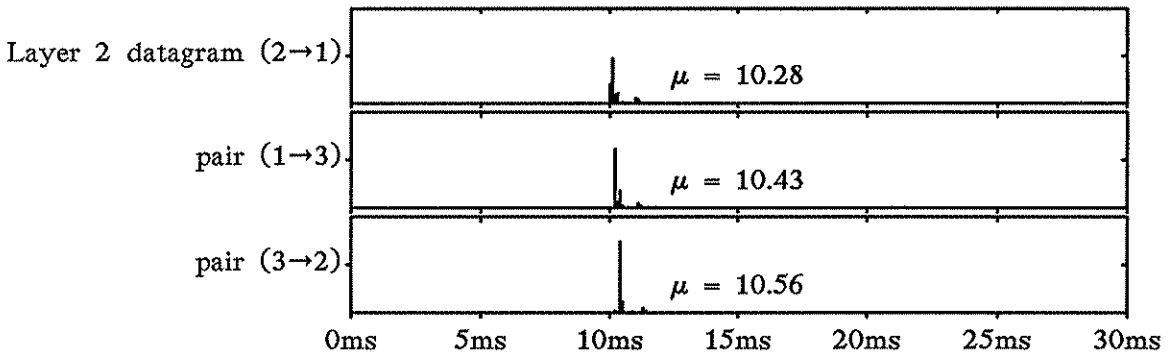
Figure 3-12 (f).

Effect of Configuration on Delay Distributions.

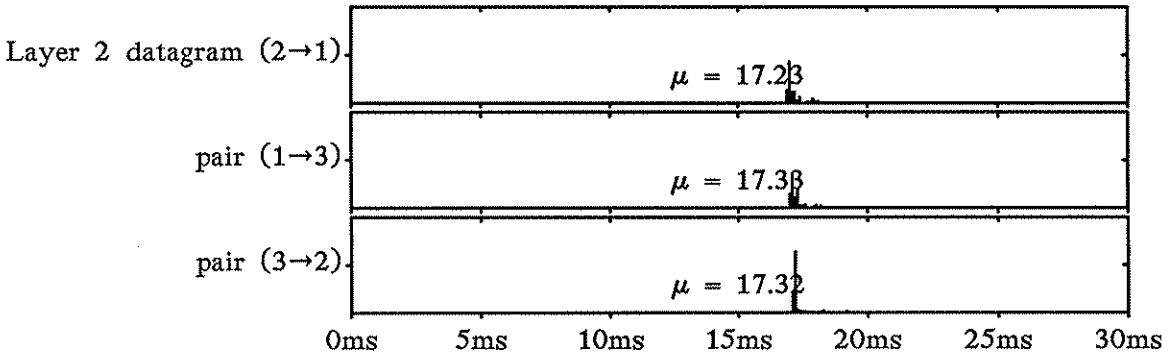
(512 Bytes/Packet)



CqCommRB Request Block Delay



RqReceiveMessage Response Delay

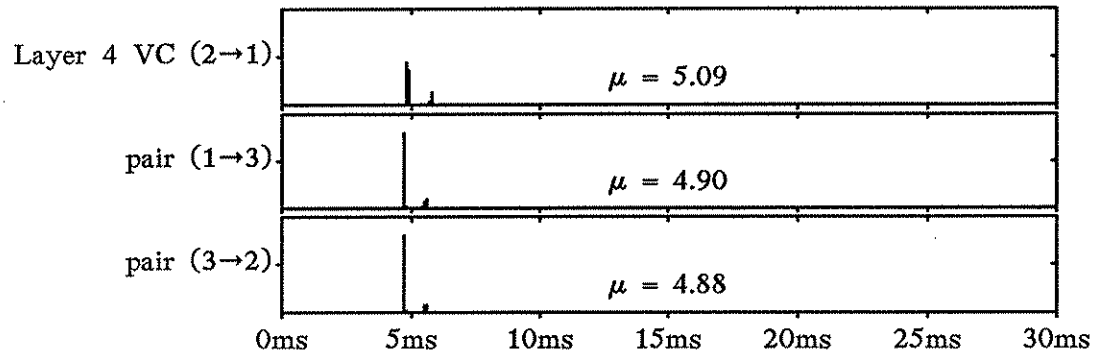


End to End Delay

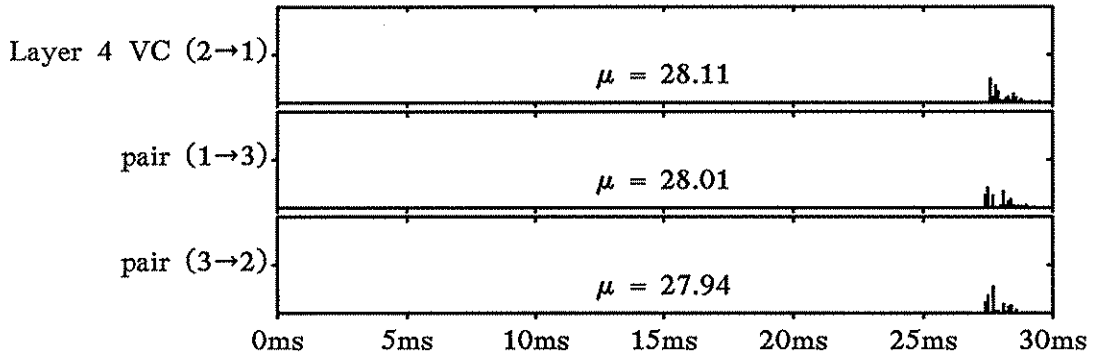
Figure 3-12 (g).

Effect of Configuration on Delay Distributions.

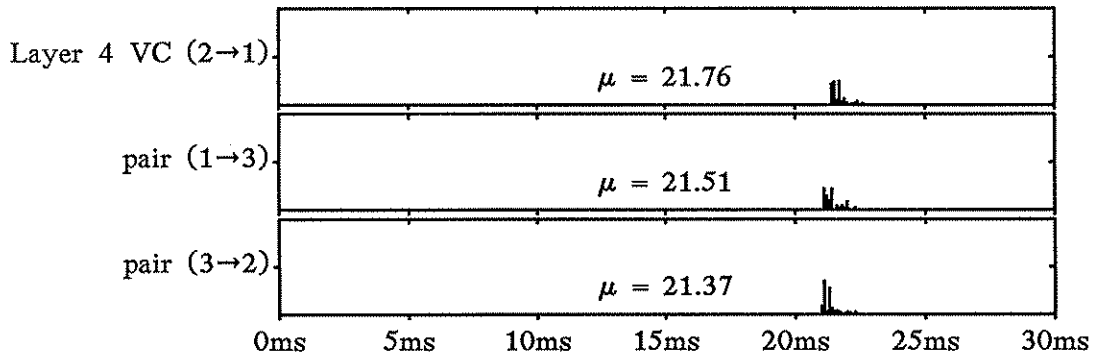
(1500 Bytes/Packet)



CqCommRB Request Block Delay



RqReceiveMessage Response Delay



End to End Delay

Figure 3-12 (h).

Effect of Configuration on Delay Distributions.

(1500 Bytes/Packet)

It is clear that the minimum overhead for the Ethernet is the 5-millisecond delay for submitting a request block. Since this is constant for all packets and constitutes a significant portion (sometimes more than half) of the total end-to-end delay, greatest throughput efficiency will be achieved using larger packet sizes.

Figure 3-7 shows that there is little reason for using the layer 4 datagram service--it is slower than the layer 2 datagram service, and is less convenient than the VC service.

Figure 3-9 indicates that although traffic on the net degrades performance somewhat, the effect is predictable. It is likely that if higher levels of traffic could have been generated, the effects would have followed the same pattern.

Figure 3-11 shows that the performance differences between machines that was sometimes noted in layer 7 experiments was not evident at the lower layers.

Chapter 4

Source Code

4.1. INC is a directory of files to be `included` in PL/M sources. Most define common data structures and external subroutine interfaces. A few small routines here are to be included directly, rather than linked in.

4.1.1. INC/DCL.LIT defines a few abbreviations familiar to XPL and PL/I programmers.

4.1.2. INC/RBHDR.INC defines the request block interface to iNA 961. The operation and return mailbox are described by the request block (`RB`), which is passed to the system routine `CqCommRB` . The header common to all request blocks is defined here.

Pointers passed to iNA 961 must be absolute addresses, unlike the PL/M `pointer` type. **INC/RBHDR.INC** also contains routines to convert pointers.

These definitions are taken from the *iNA 960-1 Programmer's Reference Manual*, especially section 2.2 and the *iNA 961 Programmer's Reference Manual*, section 3.2.

4.1.3. INC/DL.INC defines the request blocks for the Data Link layer interface (connect, disconnect, transmit, and receive operations).

4.1.4. INC/TL.INC defines the request blocks for the Transport layer interface. The datagram, open, connection, and virtual circuit request blocks formats are each used for several operations. These are taken from the *iNA 960-1 Programmer's Reference Manual*, figures 5-1 through 5-6. Also defined are a format for a Transport layer address and a format for a buffer descriptor.

4.1.5. INC/PUTBYT.INC prints a string of bytes in hexadecimal.

4.1.6. INC/ZAPBYT.INC zeros a string of bytes.

4.2. MOD is a directory of separately compiled modules used by other programs. **MOD/*.SRC** are the module sources. The external interface of each module is described by a **INC/*.EXT** file.

4.2.1. MOD/DL defines the routines `DLConnect`, `DLDisconnect`, `DLTransmit`, and `DLReceive`. Each routine requires a mailbox and an iNA 961 User ID (obtained by calling `CqCreateCommUser`). `DLConnect`, `DLDisconnect`, and `DLReceive` require a Data Link Service Access Point (`DLSAP`), much like a socket number. The `DLSAP` must be connected before use and disconnected after use.

`DLReceive` will wait up to 6000 system timer units (roughly a minute) before failing.

The public flag byte `DLTrace` controls runtime trace output from these routines.

4.2.2. MOD/VC defines routines for establishing and using transport layer virtual circuits. These require mailbox and iNA 961 User ID.

VCOpen and VCClose create and destroy a virtual circuit endpoint, identified by its reference number. VCConnect issues or awaits a connect.

PasTAB and ActTAB construct Transport Address Buffers (TABs) suitable for active open (send connect) and passive open (await connect). They return Transport Address Buffer Pointers (TAPs) suitable for VCConnect.

VCSend sends a packet. The EOM flag indicates that this is the last packet of some series. VCReceive receives a packet and returns the size. It also requires a pointer to a byte for the packet's EOM status.

VCReceive and VCConnect (passive open) will wait up to 6000 system timer units (roughly a minute) before failing.

The public flag byte VCTrace controls runtime trace output from/take these routines.

4.2.3. MOD/NMFREAD has a subroutine NMFRead to write and read the Network Management Facility statistics. This routine requires the iNA 961 User ID, the operation (read, read and write, read and clear), the object and modifier, and a pointer and length for the value. The operations, objects, and modifiers are described in the *iNA 960-1 Programmer's Reference Manual*, section 6.4 and Appendix B.

4.2.4. MOD/TICKER defines a faster clock than the second counter provided by the system. The ticker uses the 8254 clock number 2, otherwise unused by our system. The clock ticks at approximately 1.23MHz, wrapping in about 53 msec. The clock is polled every 3 system timer units (roughly 30 msec.) to detect the counter wrapping.

`TickStart` starts the clock and polling task. `Ticks` returns a doubleword which is incremented every clock tick. This wraps in a little more than 58 minutes. `TickUntil` waits until a specified tick count has been passed.

4.2.5. MOD/FMT defines the routines for formatted I/O used nearly everywhere else. These somewhat resemble the C `printf` and `scanf` routines. `ConsOpen` and `ConsClose` open and close the console for use by the job.

`ConsPut` formats data according to a format string. A pointer indicates the start of data in memory. The format string is printed, with data formatted and substituted for format characters signalled by a backtick (grave accent) character ("`").

`ConsGet` is called like `ConsPut`. It prints the format string but reads input upon encountering a format character. The pointer indicates the start of the input memory.

When a string is input it must be initialized with the maximum length in the first byte. The "`?" format character indicates that the current value is to be used as the default input. Numeric and string input is ended by a carriage return character. The last input character is deleted by the DEL character. Numeric input is echoed in a strange way.

Numbers are in decimal by default. Format characters can change the radix of the next number to 16, 10, 8, or 2. Numbers are unsigned.

`FPut` formats output to a file instead of the console.

4.2.6. MOD/EXCEPT defines a word Exc to use for exception codes for system routines, and a subroutine ExcChk to check for nonzero exception codes.

4.3. TEST is a directory of test drivers for the subroutines in **MOD**. These are useful for debugging the network interfaces. They contain examples of calls to the subroutines defined in **MOD**.

4.3.1. TEST/DLTEST can send and receive Link layer datagrams.

4.3.2. TEST/VCTEST can establish a Transport layer virtual circuit and send and receive packets.

4.3.3. TEST/NMFTEST is used to read and reset Network Management Facility statistics. It has commands to read, read and clear, and format statistics.

4.4. UTILS is a directory of utility programs. The sources are in **UTILS/*.SRC**. **UTILS/*.CSD** are submit files to compile and link the sources.

4.4.1. UTILS/JAM is used to generate bus traffic by sending Link layer datagrams. It asks for a destination network address (DA) and DLSAP, the number and size of packets to send and the number of request blocks to use. Using several request blocks allows overlap of the CqCommRB calls with I/O. Experience is that 2 RBs is usually sufficient.

4.4.2. UTILS/DG prints a few Network Management Facility statistics. These are the network address, number of packets sent, primary and secondary collisions, exceeded collisions, number of packets received, CRC errors, unaligned packets, and

receive overruns.

4.5. Source Code Listings

4.5.1. INC/DCL.LIT

```
1  declare dcl literally 'declare', /* a la XPL */
2          lit literally 'literally';
```

4.5.2. INC/RBHDR.INC

```
 1  $subtitle('Request Block Header')
 2
 3  /* avoid a type mismatch in /rmxnet/inc/cqrb.ext */
 4  CommRB: procedure(RBToken,ExcPtr);
 5      dcl RBToken token,
 6          ExcPtr pointer;
 7      dcl RBWord word at (@RBToken);
 8      call CqCommRB(RBWord,ExcPtr);
 9      end CommRB;
10
11  /* convert to absolute address */
12  AbsPtr: procedure(P) dword;
13      dcl P pointer;
14      dcl Dw dword;
15      dcl SS structure( Offs word, Sel word) at (@P);
16      Dw = SS.Sel;
17      Dw = Dw*16 + SS.Offs;
18      return Dw;
19      end AbsPtr;
20
21  /* according to iNA960-1 P.R.M. section 2.2.2 */
22  /* but see also AbsPtr P.R.M. section 3.2 */
23
24  dcl RBHdrSel selector, /* to RBHdr */
25      RBHdr based RBHdrSel structure
26        ( Rsvd (2) word,
27          Length byte,
28          User word,
29          RespPort byte,
30          RetMbx token, /* changed in AbsPtr */
31          RBSel token, /* changed in AbsPtr */
32          SubSys byte,
33          Op byte,
34          Resp word ),
35      RBHdrSize lit '16';
```

4.5.3. INC/DL.INC

```
$subtitle('Data Link requests')

/* connect RB */
declare CRBSel selector,
  CRB based CrbSel structure
   ( Hdr (RBHdrSize) byte,
     Dlsap    byte,
     Reserved byte,
     Port     byte );

/* disconnect RB */
declare DRBSel selector,
  DRB based DrbSel structure
   ( Hdr (RBHdrSize) byte,
     Dlsap    byte,
     Reserved byte );

/* transmit RB */
declare TRBSel selector,
  TRB based TrbSel structure
    ( Hdr (RBHdrSize) byte,
      Reserved word,
      BufCount word,
      ByteCount  (4) word,
      BufLoc     (4) dword,
      DstAddrPtr  dword );

/* receive RB */
declare RRBSel selector,
     RRB based RRBSel structure
    ( Hdr (RBHdrSize) byte,
      Dlsap  byte,
      Reserved  byte,
      BufCount  word,
      RetCount  word,
      ByteCount  (4) word,
      BufLoc     (4) dword );
```

4.5.4. INC/TL.INC

```
$subtitle ('Transport Layer RBs')

/* Transport address buffer (figure 5-1) */
dcl TABPtr pointer,
    TAB based TABPtr structure
      ( MBZ byte, /* must be zero */
        LocTSAPLen byte, /* local TSAP ID length = 2 */
        LocTSAP (2) byte, /* local TSAP ID */
        RemNetLen byte, /* remote net address length = 12 */
        RemNet (12) byte, /* remote net address */
        RemTSAPLen byte, /* remote TSAP ID length = 2 */
        RemTSAP (2) byte ), /* remote TSAP ID */
    TABSize lit '20'; /* size of this mess */

/* buffer descriptor (figure 5-2) */
dcl BlocksPtr pointer,
    Blocks based BlocksPtr (1) structure /* array of blocks */
      ( Addr dword, /* iNA961 address */
        Length word ),
    BlockSize lit '6'; /* size of one descriptor block */

/* Datagram RB (figure 5-6) */
dcl DGRBSel selector,
    DGRB based DGRBSel structure
      ( Hdr (RBHdrSize) byte,
        Rsvd (4) byte,
        TAAddr dword, /* TA buffer iNA961 address */
        TSAPClass byte,
        BufLen word,
        NumBlks byte, /* start of buffer descriptor */
        Blocks (BlockSize) byte ); /* room for 1 block */

 /* Open RB (figure 5-3) */
dcl ORBSel selector,
    ORB based ORBSel structure
      ( Hdr (RBHdrSize) byte,
        Reference word ); /* connection reference */

/* VC RB (figure 5-5) */
dcl VCRBSel selector,
    VCRB based VCRBSel structure
      ( Hdr (RBHdrSize) byte,
        Reason byte, /* ISO reason code */
        Rsvd (15) byte,
        Reference word,
        ConnClass byte,
        BufLen word,
        NumBlks byte,
        Blocks (BlockSize) byte ); /* room for 1 block */

/* Connection RB (figure 5-4) */
dcl CRBSel selector,
    CRB based CRBSel structure
      ( Hdr (RBHdrSize) byte,
        Reason byte, /* ISO reason code */
        Rsvd (4) byte,
        AckDelay word, /* ACK Delay estimate */
        TAAddr dword, /* TA buffer iNA961 address */
        Persist word, /* persistence count */
        Timeout word, /* abort timeout */
        Reference word,
        ConnClass word, /* connection class */
        Options word, /* negotiation options */
        DataOffset word, /* user data buffer offset */
```

```
65          DataBase word, /* & base */
66          DataLength word ); /* & length */
```

4.5.5. MOD/DL.SRC

```
$title('DataLink')
$large
$optimize(3)

/* lots of code from "/user/robert/dl*.src" */

DL: do;

$include(:me:inc/dcl.lit)
$include(:sd:rmx86/inc/ltksel.lit)

$include(:sd:rmx86/inc/ncrseg.ext)
$include(:sd:rmx86/inc/ndlseg.ext)
$include(:sd:rmx86/inc/nrcmes.ext)

$include(:sd:rmxnet/inc/cqrb.ext)

$include(:me:inc/fmt.ext)
$include(:me:inc/except.ext)
$include(:me:inc/rbhdr.inc)
$include(:me:inc/dl.inc)
$include(:me:inc/zapbyt.inc)
$include(:me:inc/putbyt.inc)

```

```
25  $subtitle('DataLink interface')
26
27  dcl DLTrace byte public initial (0); /* debugging flag */
28
29  /* Transmit packet header (P.R.M. section 3.4.2) */
30  dcl THdrPtr pointer,
31      THdr based THdrPtr structure
32        ( DstLSAP byte,
33          SrcLSAP byte,
34          ISOCmd byte), /* must be 03H */
35      THdrSize lit '3'; /* size of header */
36
37  /* Receive packet header (P.R.M. section 3.4.4) */
38  dcl RHdrPtr pointer,
39      RHdr based RHdrPtr structure
40        ( DA (6) byte, /* network destination */
41          SA (6) byte, /* network source */
42          InfoLen word, /* length of packet */
43          DstLSAP byte, /* destination AP */
44          SrcLSAP byte, /* source AP */
45          ISOCmd byte), /* should be 03H */
46      RHdrSize lit '17'; /* size of header */
47
```

```
48  $eject
49
50  /* create a RB and initialize the header */
51  CreateRB: procedure (Mbx,User,Len,Op) selector;
52      dcl Mbx token,
53          User word, /* job ID */
54          Len byte, /* RB length */
55          Op byte; /* op code */
56      RBHdrSel = RqCreateSegment(Len,@Exc); call ExcChk; /* get */
57      call ZapBytes(@RBHdr,Len); /* clear */
58      RBHdr.Length = Len; /* stuff */
59      RBHdr.User = User;
60      RBHdr.RespPort = 0FFH;
61      RBHdr.RetMbx = Mbx;
62      RBHdr.RBSel = RBHdrSel;
63      RBHdr.SubSys = 20H;
64      RBHdr.Op = Op;
65      return RBHdrSel;
66      end CreateRB;
67
68  /* issue a request and get the response */
69  UseRB: procedure (Mbx,Sel);
70      dcl Mbx token,
71          Sel selector; /* RB */
72      dcl Resp word; /* mailbox response */
73
74      RBHdrSel = Sel;
75      RBHdr.Resp = 0; /* just to make sure */
76      if DLTrace<>0
77      then call DumpRB;
78      call CommRB(Sel,@Exc); call ExcChk;
79      Sel = RqReceiveMessage(Mbx,6000,@Resp,@Exc); call ExcChk;
80      if (Resp<>0) or (Sel<>RBHdrSel) or (RBHdr.Resp<>1)
81      then do; Exc = 0FFF0H; call ExcChk; end;
82      end UseRB;
83
84  /* tracing aid */
85  DumpRB: procedure;
86      call ConsPut(@RbHdrSel,@(''↑DL use RB @ 'F'W : '.'));
87      call PutBytes(@RBHdr,RBHdr.Length);
88      call ConsPut(nil,@(''↑'.'));
89      end DumpRB;
90
```

```
 91  $eject
 92
 93  /* connect */
 94  DLConnect: procedure(Mbx,User,DLSAP) public;
 95      dcl Mbx token,
 96          User word, /* job ID */
 97          DLSAP byte; /* access point */
 98      CRBSel = CreateRB(Mbx,User,Size(CRB),82H);
 99      if DLTrace<>0
100      then call ConsPut(@DLSAP,@(''↑DL connect DSAP='B'↑'.'));
101      CRB.DLSAP = DLSAP; CRB.Port = 0FFH;
102      call UseRB(Mbx,CRBSel);
103      call RqDeleteSegment(CRBSel,@Exc); call ExcChk;
104      end DLConnect;
105
106  /* disconnect */
107  DLDisconnect: procedure(Mbx,User,DLSAP) public;
108      dcl Mbx token,
109          User word, /* job ID */
110          DLSAP byte; /* access  point */
111      DRBSel = CreateRB(Mbx,User,Size(DRB),83H);
112      DRB.DLSAP = DLSAP;
113      if DLTrace<>0
114      then call ConsPut(@DLSAP,@(''↑DL disconnect DSAP='B'↑'.'));
115      call UseRB(Mbx,DRBSel);
116      call RqDeleteSegment(DRBSel,@Exc); call ExcChk;
117      end DLDisconnect;
118
```

```
119  $eject
120
121  /* transmit */
122  DLTransmit: procedure (Mbx,User,DAP,PktPtr,PktSize) public;
123      dcl Mbx token,
124          User word, /* job ID */
125          DAP pointer, /* to network address */
126          PktPtr pointer, /* for packet */
127          PktSize word; /* of packet */
128
129      TRBSel = CreateRB(Mbx,User,Size(TRB),84H);
130      TRB.BufCount = 1;
131      TRB.ByteCount(0) = PktSize;
132      TRB.BufLoc(0) = AbsPtr(PktPtr);
133      TRB.DstAddrPtr = AbsPtr(DAP);
134      if DLTrace<>0
135      then do;
136          call ConsPut(DAP,@(''↑DL xmit DA='F'B'F'B'F'B'F'B'F'B'F'B '.'));
137          call ConsPut(@PktPtr,@('Pkt @'F'P'.'));
138          call ConsPut(@PktSize,@('('W)'↑'.'));
139          end;
140      call UseRB(Mbx,TRBSel);
141      call RqDeleteSegment(TRBSel,@Exc); call ExcChk;
142      end DLTransmit;
143
144  /* receive */
145  DLReceive: procedure (Mbx,User,DLSAP,PktPtr,PktSize) public;
146      dcl Mbx token,
147          User word, /* job ID? */
148          DLSAP byte, /* access point */
149          PktPtr pointer, /* to packet */
150          PktSize word; /* of packet */
151
152      RRBSel = CreateRB(Mbx,User,Size(RRB),85H);
153      RRB.DLSAP = DLSAP;
154      RRB.BufCount = 1;
155      RRB.ByteCount(0) = PktSize;
156      RRB.BufLoc(0) = AbsPtr(PktPtr);
157      call UseRB(Mbx,RRBSel);
158      if DLTrace<>0
159      then do;
160          call ConsPut(nil,@(''↑DL recv '.'));
161          call ConsPut(@PktPtr,@('Pkt @'F'P'.'));
162          call ConsPut(@PktSize,@('('W)'↑'.'));
163          end;
164      call RqDeleteSegment(RRBSel,@Exc); call ExcChk;
165      end DLReceive;
166  end DL;
```

4.5.6. MOD/VC.SRC

```
$title('Virtual Circuits')
$large
$optimize(3)

VC: do;

$include(:me:inc/dcl.lit)
$include(:sd:rmx86/inc/ltksel.lit)

$include(:sd:rmx86/inc/ncrseg.ext)
$include(:sd:rmx86/inc/ndlseg.ext)
$include(:sd:rmx86/inc/nrcmes.ext)

$include(:sd:rmxnet/inc/cqrb.ext)
$include(:sd:rmxnet/inc/cqtl.lit)

$include(:me:inc/fmt.ext)
$include(:me:inc/except.ext)
$include(:me:inc/rbhdr.inc)
$include(:me:inc/tl.inc)
$include(:me:inc/zapbyt.inc)
$include(:me:inc/putbyt.inc)

```

```
24  $subtitle('VC interface')
25
26  dcl VCTrace byte public initial (0); /* debugging flag */
27
28  /* create a RB and initialize the header */
29  CreateRB: procedure (Mbx,User,Len,Op) selector;
30      dcl Mbx token,
31          User word, /* job ID */
32          Len byte, /* RB length */
33          Op byte; /* op code */
34      RBHdrSel = RqCreateSegment(Len,@Exc); call ExcChk; /* get */
35      call ZapBytes(@RBHdr,Len); /* clear */
36      RBHdr.Length = Len; /* stuff */
37      RBHdr.User = User;
38      RBHdr.RespPort = 0FFH;
39      RBHdr.RetMbx = Mbx;
40      RBHdr.RBSel = RBHdrSel;
41      RBHdr.SubSys = 40H;
42      RBHdr.Op = Op;
43      return RBHdrSel;
44      end CreateRB;
45
46  /* issue a request and get the response */
47  UseRB: procedure (Mbx,Sel);
48      dcl Mbx token,
49          Sel selector, /* RB */
50          Resp word; /* Mbx resp */
51
52      RBHdrSel = Sel;
53      RBHdr.Resp = 0; /* just to make sure */
54      if VCTrace<>0
55      then call DumpRB;
56      call CommRB(Sel,@Exc); call ExcChk;
57      Sel = RqReceiveMessage(Mbx,6000,@Resp,@Exc); call ExcChk;
58      if (Resp<>0) or (Sel<>RBHdrSel) or ((RBHdr.Resp and 0001H)=0)
59      then do; Exc = 0FF80H; call ExcChk; end;
60      end UseRB;
61
62  /* tracing aid */
63  DumpRB: procedure;
64      call ConsPut(@RbHdrSel,@(''↑VC use RB @ 'F'W : '.'));
65      call PutBytes(@RBHdr,RBHdr.Length);
66      call ConsPut(nil,@(''↑'.'));
67      end DumpRB;
```

```
68  $eject
69
70  /* create a socket and return its reference number */
71  VCOpen: procedure(Mbx,User) word public;
72      dcl Mbx token,
73          User word;
74      dcl Ref word; /* VC reference */
75
76      ORBSel = CreateRB(Mbx,User,Size(ORB),0);
77      call UseRB(Mbx,ORBSel);
78      Ref = ORB.Reference;
79      if VCTrace<>0
80      then call ConsPut(@Ref,@(''↑VC open reference 'F'W'↑'.'));
81      call RqDeleteSegment(ORBSel,@Exc); call ExcChk;
82      return Ref;
83      end VCOpen;
84
85  /* close a socket */
86  VCClose: procedure(Mbx,User,Reference) public;
87      dcl Mbx token,
88          User word,
89          Reference word;
90      VCRBSel = CreateRB(Mbx,User,Size(VCRB),CloseReq);
91      VCRB.Reference = Reference; /* default everything */
92      call UseRB(Mbx,VCRBSel);
93      if VCTRace<>0
94      then call ConsPut(@Reference,@(''↑VC close reference 'F'W'↑'.'));
95      call RqDeleteSegment(VCRBSel,@Exc); call ExcChk;
96      end VCClose;
97
```

```
 98  $eject
 99
100  /* create passive connect Transport Address Buffer */
101  PasTAB: procedure (LocTSAP,RemNetPtr,RemTSAP) pointer public;
102      dcl LocTSAP word, /* local AP */
103          RemNetPtr pointer, /* to remote net address */
104          RemTSAP word; /* remote node AP */
105      dcl TABSel selector,
106          TA based TABSel (TABSize) byte;
107
108      TABSel = RqCreateSegment(TABSize,@Exc); call ExcChk;
109      TABPtr = @TA;
110      call ZapBytes(TABPtr,TABSize);
111      TAB.LocTSAPLen = 2; call MovB(@LocTSAP,@TAB.LocTSAP,2);
112      TAB.RemTSAPLen = 2; call MovB(@RemTSAP,@TAB.RemTSAP,2);
113      TAB.RemNetLen = 12; call MovB(RemNetPtr,@TAB.RemNet(4),6);
114      return TABPtr;
115      end PasTAB;
116
117  /* create active connect TAB */
118  ActTAB: procedure (LocTSAP,RemNetPtr,RemTSAP) pointer public;
119      dcl LocTSAP word, /* local AP */
120          RemNetPtr pointer, /* to remote net address */
121          RemTSAP word; /* remote node AP */
122      TABPtr = PasTAB(LocTSAP,RemNetPtr,RemTSAP);
123      TAB.RemNet(3), TAB.RemNet(11) = 1;
124      return TABPtr;
125      end ActTAB;
126
127  /* active/passive connect to TAP socket.  No data or options */
128  VCConnect: procedure(Mbx,User,Reference,TAP,Act) public;
129      dcl Mbx token,
130          User word,
131          Reference word,
132          TAP pointer, /* to Transport Address buffer */
133          Act byte; /* active open flag */
134      dcl Op byte;
135
136      if Act<>0 then Op = SendConnReq; else Op = AwaitConnReqTran;
137      CRBSel = CreateRB(Mbx,User,Size(CRB),Op);
138      CRB.TAAddr = AbsPtr(TAP);
139      CRB.Reference = Reference;
140      call UseRB(Mbx,CRBSel);
141      if VCTrace<>0
142      then do;
143          call ConsPut(@Op,@(''↑VC connect 'F'B '.'));
144          call ConsPut(@Reference,@('reference 'F'W'↑TA '.'));
145          call PutBytes(TAP,TABSize);
146          call ConsPut(Nil,@(''↑'.'));
147          end;
148      call RqDeleteSegment(CRBSel,@Exc); call ExcChk;
149      end VCConnect;
```

```
150  $eject
151
152  /* send a packet */
153  VCSend: procedure(Mbx,User,Reference,PktPtr,PktSize,EOM) public;
154      dcl Mbx token,
155          User word,
156          Reference word,
157          PktPtr pointer, /* to data */
158          PktSize word, /* of data */
159          EOM byte; /* if end of unit */
160      dcl Op byte;
161
162      if EOM=0 then Op = SendData; else Op = SendEOMData;
163      VCRBSel = CreateRB(Mbx,User,Size(VCRB),Op);
164      VCRB.Reference = Reference;
165      VCRB.NumBlks = 1;
166      BlocksPtr = @VCRB.Blocks;
167      Blocks(0).Addr = AbsPtr(PktPtr);
168      Blocks(0).Length = PktSize;
169      call UseRB(Mbx,VCRBSel);
170      if VCTrace<>0
171      then do;
172          call ConsPut(@Op,@(''tVC send 'F'B '.'));
173          call ConsPut(@Reference,@('reference 'F'W '.'));
174          call ConsPut(@PktPtr,@('@ 'F'P '.'));
175          call ConsPut(@PktSize,@('('W)'t'.'));
176          end;
177      call RqDeleteSegment(VCRBSel,@Exc); call ExcChk;
178      end VCSend;
179
180  /* receive a packet.  Set the EOM status and return the length */
181  VCReceive: procedure(Mbx,User,Reference,PktPtr,PktSize,EOMPtr) word public;
182      dcl Mbx token,
183          User word,
184          Reference word,
185          PktPtr pointer, /* to data */
186          PktSize word, /* of data */
187          EOMPtr pointer; /* to EOM byte flag */
188      dcl EOM based EOMPtr byte, /* if end of unit */
189          Rcvd word; /* bytes received */
190
191      VCRBSel = CreateRB(Mbx,User,Size(VCRB),ReceiveData);
192      VCRB.Reference = Reference;
193      VCRB.NumBlks = 1;
194      BlocksPtr = @VCRB.Blocks;
195      Blocks(0).Addr = AbsPtr(PktPtr);
196      Blocks(0).Length = PktSize;
197      call UseRB(Mbx,VCRBSel);
198      EOM = (RBHdr.Resp=OKEOMResp);
199      Rcvd = VCRB.BufLen;
200      if VCTrace<>0
201      then do;
202          call ConsPut(@Reference,@(''tVC receive reference 'F'W '.'));
203          call ConsPut(@PktPtr,@('@ 'F'P '.'));
204          call ConsPut(@Rcvd,@('('W) '.'));
205          call ConsPut(@EOM,@('EOM 'F'B't'.'));
206          end;
207      call RqDeleteSegment(VCRBSel,@Exc); call ExcChk;
208      return Rcvd;
209      end VCReceive;
210
211  end VC;
```

4.5.7. MOD/NMFREAD.SRC

```
$title('NMF object read/reset')
$large
$optimize(3)

NMF: do;

$include(:me:inc/dcl.lit)
$include(:sd:rmx86/inc/itksel.lit)
$include(:sd:rmx86/inc/ualloc.ext)
$include(:sd:rmx86/inc/ncrmbx.ext)
$include(:sd:rmx86/inc/ufree.ext)
$include(:sd:rmx86/inc/ndlmbx.ext)
$include(:sd:rmx86/inc/nrcmes.ext)
$include(:sd:rmxnet/inc/cqrb.ext)
$include(:me:inc/rbhdr.inc)

dcl ABSel selector, /* according to iNA960-1 P.R.M. section 6.4 */
    AB based ABSel structure
      ( Hdr (RBHdrSize) byte,
        TransAdrPtr dword,
        FillLength word,
        RespBufPtr dword,
        RespBufLength word,
        CmdBufPtr dword,
        CmdBufLength word );

dcl CmdSel selector,
    Cmd based CmdSel structure /* iNA960-1 P.R.M. section 6.4.4 */
      ( Number byte,
        Object word,
        Modifier word,
        Length word,
        Value (80) byte );
dcl RespSel selector,
    Resp based RespSel structure /* iNA960-1 P.R.M. section 6.4.5 */
      ( Number byte, /* here Number=1 */
        Object word,
        Modifier word,
        Length word,
        Value (80) byte );
```

```
41  $eject
42
43  NMFRead: procedure(User,Op,Object,Modif,Length,ValuePtr,ExcPtr) word public;
44      dcl User word,
45          Op byte, /* operation code */
46          Object word, /* object */
47          Modif word, /* modifier */
48          Length word, /* length of object */
49          ValuePtr pointer, /* address of object data */
50          ExcPtr pointer; /* exception code */
51      dcl Exc based ExcPtr word, /* exception code */
52          ObjSize word, /* size of objects */
53          Mbx token, /* response mailbox */
54          MbxResp token, /* mailbox response code */
55          FillLength word; /* length filled back */
56
57      Mbx = RqCreateMailbox(0,ExcPtr); if Exc<>0 then return 0;
58
59      ObjSize = Size(Cmd)-Size(Cmd.Value)+Length;
60
61      CmdSel = DqAllocate(ObjSize,ExcPtr); if Exc<>0 then return 0;
62      Cmd.Number = 1;
63      Cmd.Object = Object;
64      Cmd.Modifier = Modif;
65      Cmd.Length = Length;
66      call MovB(ValuePtr,@Cmd.Value,Length);
67
68      RespSel = DqAllocate(ObjSize,ExcPtr); if Exc<>0 then return 0;
69
70      ABSel = DqAllocate(Size(AB),ExcPtr); if Exc<>0 then return 0;
71
72      AB.TransAdrPtr = 0; /* local node */
73      AB.RespBufPtr = AbsPtr(@Resp); /* response buffer */
74      AB.RespBufLength = ObjSize;
75      AB.CmdBufPtr = AbsPtr(@Cmd); /* command buffer */
76      AB.CmdBufLength = ObjSize;
77
78      RBHdrSel = ABSel;
79      RBHdr.Length=Size(AB);
80      RBHdr.User = User;
81      RBHdr.RespPort = 0FFh; /* for iRMX-86 */
82      RBHdr.RetMbx = Mbx;
83      RBHdr.RBSel = ABSel;
84      RBHdr.SubSys = 80h; /* network management facility */
85      RBHdr.Op = Op; /* operation */
86      RBHdr.Resp = 0; /* nothing yet */
87
88      call CommRB(ABSel,ExcPtr); if Exc<>0 then return 0;
89      ABSel = RqReceiveMessage(Mbx,1000,@MbxResp,ExcPtr); if Exc<>0 then return 0;
90      RBHdrSel = ABSel;
91      if RBHdr.Resp<>1 then return 0;
92
93      call MovB(@Resp.Value,ValuePtr,Length);
94      FillLength = AB.FillLength;
95
96      call RqDeleteMailbox(Mbx,ExcPtr); if Exc<>0 then return 0;
97      call DqFree(CmdSel,ExcPtr); if Exc<>0 then return 0;
98      call DqFree(RespSel,ExcPtr); if Exc<>0 then return 0;
99      call DqFree(ABSel,ExcPtr); if Exc<>0 then return 0;
100
101     return FillLength;
102     end NMFRead;
103 end NMF;
```

4.5.8. MOD/TICKER.SRC

```
$title('ticker — timing module')
$large
$optimize(3)

/* Ticker
 * The Ticks() function returns a dword which increments at the
 * 8254 clock rate of 1.23MHz.
 * Call TicksStart to fire it up.
 * 8254 code taken from Robert Simoncic.
 * Tasking from "Programming Techniques", ch. 3.
 */

Ticker: do;

$include(:me:inc/dcl.lit)
$include(:sd:rmx86/inc/ltksel.lit)
$include(:sd:rmx86/inc/ncrreg.ext)
$include(:sd:rmx86/inc/ncrtsk.ext)
$include(:sd:rmx86/inc/nrcctl.ext)
$include(:sd:rmx86/inc/nsnctl.ext)
$include(:sd:rmx86/inc/ndlreg.ext)
$include(:sd:rmx86/inc/ndltsk.ext)
$include(:sd:rmx86/inc/nsleep.ext)
$include(:me:inc/except.ext)
$include(:me:inc/fmt.ext)

dcl ClockN lit '2'; /* clock number */

dcl ClockAdr(4) word data (00D0H, 00D2h, 00D4h, 00D6h);
dcl ClockReg token, /* access to clock */
    ClockExc word; /* exception in region ops */

dcl LastTime dword public, /* last value returned */
    Last structure (LSW word, MSW word) at (@LastTime); /* in words */

dcl PollTask token, /* clock wrap polling */
    PollPri lit '0', /* polling priority */
    PollTime lit '3'; /* sampling interval */

/* initialize the clock */

ClockSet: procedure;
    dcl Ctl lit '0B4H', /* control byte */
        LSB lit '0FFH', /* reset value */
        MSB lit '0FFH';

    output(ClockAdr(3)) = Ctl;
    output(ClockAdr(ClockN)) = LSB;
    output(ClockAdr(ClockN)) = MSB;
    end ClockSet;

/* read the clock */

ClockRead: procedure word reentrant;
    dcl Ctl lit '80H'; /* control byte */
    dcl CW word, /* clock word */
        C structure (LSB byte, MSB byte) at (@CW); /* in bytes */

    output(ClockAdr(3)) = Ctl;
    C.LSB = input(ClockAdr(ClockN));
    C.MSB = input(ClockAdr(ClockN));
    return CW;
    end ClockRead;

```

```
65  /* start up the polling task */
66
67  PollStart: procedure;
68      dcl DataSel selector; /* our own data segment */
69
70      DataSel = SelectorOf(@DataSel);
71      PollTask = RqCreateTask(PollPri,@Poll,DataSel,0,512,0,@Exc);
72      call ExcChk;
73      end PollStart;
74
75  /* tickle the clock often enough to detect wrapping */
76
77  Poll: procedure;
78      do while 0FFH;
79          if 0<>Ticks then;
80          call RqSleep(PollTime,@Exc); call ExcChk;
81          end;
82      end Poll;
83
84  /* Get the current time.  If it's not later then it wrapped */
85
86  Ticks: procedure dword reentrant public;
87      dcl C word, /* current clock */
88          T dword; /* local copy of current ticker */
89
90      call RqReceiveControl(ClockReg,@ClockExc); /* sieze control */
91      C = -ClockRead; /* get (incrementing) LSW */
92      do while C<=Last.LSW; /* non-increasing */
93          Last.LSW = C; Last.MSW = Last.MSW+1; /* wrap it */
94          C = -ClockRead; /* try again */
95          end;
96      Last.LSW = C; /* advance LSW */
97      T = LastTime; /* get local copy */
98      call RqSendControl(@ClockExc); /* and release control */
99      return T;
100     end Ticks;
101
102 /* start everything up */
103
104 TickStart: procedure public;
105     LastTime = 0;
106     ClockReg = RqCreateRegion(1,@Exc); call ExcChk;
107     call ClockSet;
108     call PollStart;
109     end TickStart;
110
111 /* wait until a time, then return the current time */
112
113 TickUntil: procedure (Until) dword reentrant public;
114     dcl Until dword; /* until then */
115     dcl Now dword;
116     Again:
117         Now = Ticks;
118         if Now<Until then goto Again;
119     return Now;
120     end TickUntil;
121
122 end Ticker;
```

4.5.9. MOD/FMT.SRC

```
$title('NonStandard I/O')
$optimize(3)
$large

/*************************************************************/
/* formatted I/O.  The format string contains text mixed     */
/* with the formats:                                         */
/*     '.  end of string (required)                          */
/*     ''  prints "'"                                        */
/*     'n  prints new line                                   */
/*     'C  a character                                       */
/*     'S  a string                                          */
/* numbers may be preceded by a radix (default decimal)      */
/*     'B  a byte          '9 use decimal                    */
/*     'W  a word          'F use hex                        */
/*     'D  a double word   '7 use octal                      */
/*     'P  a pointer       '1 use binary                     */
/*     'X  skip a byte                                       */
/*     '?  use current value as default for input            */
/*************************************************************/

Fmt: do;

/* things we might want to know about */
$save nolist
$include(:me:inc/dcl.lit)
$include(:sd:rmx86/inc/ltksel.lit)
$include(:sd:rmx86/inc/ucreat.ext)
$include(:sd:rmx86/inc/uclose.ext)
$include(:sd:rmx86/inc/uwrite.ext)
$include(:sd:rmx86/inc/uread.ext)
$include(:sd:rmx86/inc/uspecl.ext)
$include(:sd:rmx86/inc/uopen.ext)
$restore
```

```
35  $eject
36
37  dcl ConsOut token public, /* :CO: token */
38      ConsIn token public; /* :CI: token */
39
40  /* hook up a console connection */
41  ConsOpen: procedure public;
42      ConsOut = DqCreate(@(4,':CO:'),@FmtExc);
43      call DqOpen(ConsOut,2,0,@FmtExc);
44      ConsIn = DqCreate(@(4,':CI:'),@FmtExc);
45      call DqOpen(ConsIn,1,0,@FmtExc);
46      end ConsOpen;
47
48  /* call to shut down console connection */
49  ConsClose: procedure public;
50      call DqClose(ConsOut,@FmtExc);
51      call DqClose(ConsIn,@FmtExc);
52      end ConsClose;
```

```
53  $eject
54
55  dcl FmtScan pointer public, /* current scanning pointer */
56      FmtExc token public; /* exception */
57
58  dcl FmtIn token, /* input, iff exists */
59      FmtOut token, /* output, iff exists */
60      FmtWidth word; /* output field width */
61
62  /* our read */
63  Read: procedure(P,L);
64      dcl P pointer, /* to text */
65          L byte; /* length */
66      call DqSpecial(1,@FmtIn,@FmtExc); /* set transparent */
67      if L<>DqRead(FmtIn,P,L,@FmtExc)
68      then /*??*/;
69      call DqSpecial(0,@FmtIn,@FmtExc); /* set normal */
70      end Read;
71
72  /* our write */
73  Write: procedure(P,L);
74      dcl P pointer, /* to text */
75          L word; /* length */
76      if FmtOut<>0
77      then call DqWrite(FmtOut,P,L,@FmtExc);
78      FmtWidth = FmtWidth+L;
79      end Write;
80
81  /* bump the FmtScan scanner */
82  XFmt: procedure(L);
83      dcl L word; /* offset */
84      dcl B based FmtScan (*) byte;
85      FmtScan = @B(L);
86      end XFmt;
87
88  dcl AscBEL lit '07h',
89      AscBS  lit '08h',
90      AscLF  lit '0Ah',
91      AscCR  lit '0Dh',
92      AscDEL lit '07Fh';
93
94  dcl EOL (*) byte data (AscCR,AscLF), /* newline */
95      BS (32) byte
96          data  ( 8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,
97                  8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8 ),
98      Zap (*) byte data (AscBS,' ',AscBS);
99
```

```
100  $eject
101
102  dcl FmtDft byte initial (0); /* allow default input */
103
104  dcl Digit (16) byte data ('0123456789ABCDEF'), /* map to characters */
105      NSize (3) byte data (1,2,4); /* width of numbers */
106
107  dcl Rdx10 (*) byte data (10,' ',' ',3,5,10),
108      Rdx16 (*) byte data (16,'H','0',2,4,8),
109      Rdx8  (*) byte data (8,'Q','0',3,6,11),
110      Rdx2  (*) byte data (2,'B','0',8,16,32),
111      RdxPtr pointer initial (@Rdx10),
112      Rdx based RdxPtr structure /* conversion table */
113        ( Rdx byte, /* divisor */
114          Tag byte, /* suffix */
115          L0 byte, /* leading 0 */
116          Digs (3) byte ); /* digits in various widths */
117
```

```
$eject

/* scan a number */
NFmt: procedure(W);
    dcl W byte; /* size index */
    dcl N dword; /* accumulator */

    /* print N */
    NOut: procedure;
        dcl R dword, /* remains of unformatted N */
            D byte, /* digit counter */
            T (33) byte; /* buffer for text */
        R = N;
        D = Rdx.Digs(W);
        T(D) = Rdx.Tag; /* explain */
        if R=0 /* special case for all 0 */
        then do; T(D-1) = '0'; D = D-1; end;
        else do while R<>0 and D<>0; /* pull LSD to MSD */
            T(D-1) = Digit(Low(R mod Rdx.Rdx));
            R = R/Rdx.Rdx;
            D = D-1;
            end;
        do while D<>0; /* leading 0s */
            T(D-1) = Rdx.L0; D = D-1;
            end;
        call Write(@T,Rdx.Digs(W)+1); /* print */
        end NOut;

    /* input, edit, & display a number */
    NIn: procedure;
        dcl C byte; /* input character */

        if FmtDft<>0 /* use previous value */
        then FmtDft = 0; /* clear flag */
        else N = 0; /* clear acc */
        call NOut; /* initial value */
        C = ' ';
        do while C<>AscCR;
            call Write(@BS,2); call Read(@C,1);
            if C=AscDEL then N = N/Rdx.Rdx;
            else if '0'<=C and C<='9' then N = N*Rdx.Rdx+C-'0';
            else if 'A'<=C and C<='F' then N = N*Rdx.Rdx+10+C-'A';
            else if 'a'<=C and C<='f' then N = N*Rdx.Rdx+10+C-'a';
            call Write(@BS,Rdx.Digs(W)-1); call NOut;
            end;
        call MovB(@N,FmtScan,NSize(W)); /* store number */
        end NIn;

    N = 0; call MovB(FmtScan,@N,NSize(W)); /* load accumulator */
    if FmtIn<>0
    then call NIn; /* input */
    else call NOut; /* dump result */
    call XFmt(NSize(W));
    Rdxptr = @Rdx10; /* reset radix */
    end NFmt;
```

```
$eject

/* scan off a character */
CFmt: procedure;
    if FmtIn<>0
    then call Read(FmtScan,1);
    call Write(FmtScan,1);
    call XFmt(1);
    end CFmt;

/* scan off a string */
SFmt: procedure;
    dcl S based FmtScan structure
          ( L byte, /* length of string */
            B (80) byte ); /* text */

/* read into a string */
SIn: procedure;
    dcl I word, /* current length */
        C byte;
    I = 0;
    do while I<>S.L;
        call Read(@C,1);
        if C=AscCR
        then S.L=I;
        else if C=AscDEL
        then do;
            if I<>0 then do; call Write(@Zap,Length(Zap)); I = I-1; end;
            end;
        else do; call Write(@C,1);; S.B(I) = C; I = I+1; end;
        end;
    end SIn;

    if FmtIn<>0
    then call SIn;
    else call Write(@S.B,S.L);
    call XFmt(S.L); /* skip past string */
    end SFmt;
```

```
$eject

dcl FmtChars (*) byte data (''↑CSBWDPF971?X.'); /* format characters */

/* formatted I/O */
Fmt: procedure (P,A,I,O) public;
    dcl P pointer, /* to arguments */
        A pointer, /* to format */
        I token, /* input channel (0 for output) */
        O token; /* output channel */
    dcl Text based A (*) byte, /* substring */
        TextL word, /* length of substring */
        FChar byte, /* format character */
        FCase byte; /* format character number */

    FmtScan = P; FmtIn = I; FmtOut = O; /* set global registers */
    TextL = 0; /* start scan */
    do while 1; /* scan until end of formats */
        if Text(TextL)<>''' /* check for escape */
        then TextL =TextL+1; /* just include nonspecial */
        else do;
            if TextL<>0  then call Write(@Text,TextL); /* print text */
            FChar = Text(TextL+1);
            A = @Text(TextL+2); TextL = 0; /* skip ' */

            /* search format table */
            FCase = Length(FmtChars);
            do while FCase<>0 and FmtChars(FCase-1)<>FChar;
                FCase = FCase-1;
                end;
            do case FCase; /* take action */
                F0: ; /* not found */
                F1: call Write(@FChar,1); /* '' unspecial */
                F2: call Write(@EOL,Size(EOL)); /* '↑ EOL */
                F3: call CFmt; /* 'C character */
                F4: call SFmt; /* 'S string */
                F5: call NFmt(0); /* 'B byte */
                F6: call NFmt(1); /* 'W word */
                F7: call NFmt(2); /* 'D dword */
                F8: call NFmt(2); /* 'P pointer */
                F9: RdxPtr = @Rdx16; /* 'F hex */
                F10: RdxPtr = @Rdx10; /* '9 decimal */
                F11: RdxPtr = @Rdx8; /* '7 octal */
                F12: RdxPtr = @Rdx2; /* '1 binary */
                F13: FmtDft = 1; /* '? default on input */
                F14: call XFmt(1); /* 'X skip byte */
                F15: return; /* '. done */
                end;
            end;
        end;
    end Fmt;
```

```
263  $eject
264
265  /* set the FmtScan pointer & print a string */
266  ConsPut: procedure(P,A) public;
267      dcl P pointer, /* to argument */
268          A pointer; /* to text */
269      call Fmt(P,A,0,ConsOut);
270      end ConsPut;
271
272  /* set the FmtScan pointer & read a string */
273  ConsGet: procedure(P,A) public;
274      dcl P pointer, /* to argument */
275          A pointer; /* to text */
276      call Fmt(P,A,ConsIn,ConsOut);
277      end ConsGet;
278
279  FGet: procedure(P,A,I) public;
280      dcl P pointer, /* to argument */
281          A pointer, /* to text */
282          I token; /* input file */
283      call Fmt(P,A,ConsIn,0);
284      end FGet;
285
286  FPut: procedure(P,A,O) public;
287      dcl P pointer, /* to argument */
288          A pointer, /* to text */
289          O token; /* output file */
290      call Fmt(P,A,0,O);
291      end FPut;
292  end Fmt;
```

4.5.10. MOD/EXCEPT.SRC

```
$title('exception checker')
$large

Except: do; /* this is probably the wrong way to do this */
$include(:me:inc/dcl.lit)
$include(:sd:rmx86/inc/ltksel.lit)
$include(:sd:rmx86/inc/uexit.ext)
$include(:me:inc/fmt.ext)

dcl Exc word public; /* everybody's exception */

ExcChk: procedure public;
    dcl C byte;
    if Exc<>0
    then do;
        call ConsPut(@Exc,@(''↑Exception 'F'W:'.'));
        call ConsGet(@C,@(' continue (y|n) 'C'↑'.'));
        if C='n'
        then call DqExit(3); /* tube */
        end;
    end ExcChk;
end Except;
```

4.5.11. TEST/DLTEST.SRC

```
 1  $title('drive DL')
 2  $large
 3
 4  DLTest: do;
 5
 6  $include(:me:inc/dcl.lit)
 7  $include(:sd:rmx86/inc/ltksel.lit)
 8
 9  $include(:sd:rmx86/inc/ncrseg.ext)
10  $include(:sd:rmx86/inc/ndlseg.ext)
11  $include(:sd:rmx86/inc/ncrmbx.ext)
12  $include(:sd:rmx86/inc/ndlmbx.ext)
13  $include(:sd:rmx86/inc/nrcmes.ext)
14
15  $include(:sd:rmx86/inc/uexit.ext)
16
17  $include(:sd:rmxnet/inc/cqrb.ext)
18
19  $include(:me:inc/fmt.ext)
20  $include(:me:inc/except.ext)
21  $include(:me:inc/dl.ext)
22  $include(:me:inc/putbyt.inc)
23  $include(:me:inc/zapbyt.inc)
```

```
24  $subtitle('DL test')
25
26  dcl Mbx token, /* request mailbox */
27      User word; /* job ID? */
28
29  dcl TPkt structure
30        ( Hdr (THdrSize) byte, Space (1500) byte ),
31      RPkt structure
32        ( Hdr (RHdrSize) byte, Space (1500) byte ),
33      DLSAP byte, /* to test on */
34      DSize word,
35      DA (6) byte initial (0FFH,0FFH,0FFH,0FFH,0FFH,0FFH),
36      C byte,
37      I word;
38
39  call ConsOpen;
40
41  Mbx = RqCreateMailbox(0,@Exc); call ExcChk;
42  User = CqCreateCommUser(@Exc); call ExcChk;
43  call ConsGet(@DLTrace,@('trace flag = 'F'B '.'));
44  call ConsGet(@C,@('transmit/receive = ?',8,''C '.'));
45  call ConsGet(@DLSAP,@('DLSAP = 'F'B '.'));
46  DSize = 20; call ConsGet(@DSize,@('size = '?'W't'.'));
47
48  call DLConnect(Mbx,User,DLSAP);
49
50  if C='r'
51  then do;
52      call ZapBytes(@RPkt,DSize);
53      call DLReceive(Mbx,User,DLSAP,@RPkt,DSize);
54      call PutBytes(@RPkt,DSize);
55      end;
56  else if C='t'
57  then do;
58      call ConsGet(@DA,@('network DA = '?'F'B,'?'F'B,'?'F'B,'?'F'B,'?'F'B,'?'F'B't'.'));
59      THdrPtr = @TPkt;
60      THdr.DstLSAP, THdr.SrcLSAP = DLSAP;
61      THdr.ISOCmd = 03H;
62      call PutBytes(@TPkt,THdrSize);
63      I = THdrSize; C = 00H;
64      do while I<>DSize;
65          call ConsGet(@C,@(' '?'F'B'.')); TPkt.Space(I-THdrSize) = C;
66          I = I+1;
67          end;
68      call DLTransmit(Mbx,User,@DA,@TPkt,DSize);
69      end;
70  call ConsPut(nil,@(''t'.'));
71
72  call DLDisconnect(Mbx,User,DLSAP);
73
74  call ConsClose;
75  call DqExit(0);
76  end DLTest;
```

4.5.12. TEST/VCTEST.SRC

```
 1  $title('drive VC')
 2  $large
 3
 4  VCTest: do;
 5
 6  $include(:me:inc/dcl.lit)
 7  $include(:sd:rmx86/inc/ltksel.lit)
 8
 9  $include(:sd:rmx86/inc/ncrseg.ext)
10  $include(:sd:rmx86/inc/ndlseg.ext)
11  $include(:sd:rmx86/inc/ncrmbx.ext)
12  $include(:sd:rmx86/inc/ndlmbx.ext)
13  $include(:sd:rmx86/inc/nrcmes.ext)
14
15  $include(:sd:rmx86/inc/uexit.ext)
16
17  $include(:sd:rmxnet/inc/cqrb.ext)
18
19  $include(:me:inc/fmt.ext)
20  $include(:me:inc/except.ext)
21  $include(:me:inc/vc.ext)
22  $include(:me:inc/putbyt.inc)
23  $include(:me:inc/zapbyt.inc)
```

```
24  $subtitle('VC test')
25
26  dcl Mbx token,  /* request mailbox */
27      User word;  /* job ID? */
28
29  dcl Pkt (1486) byte,    /* raw data */
30      PktSize word,   /* packet size */
31      TSAP (2) word,  /* transport service access points */
32      NA (6) byte,    /* network address */
33      TAP pointer,    /* to transport address buffer */
34      Ref word,   /* socket reference */
35      EOM byte,   /* EOM flag */
36      C byte,
37      I word;
38
39  call ConsOpen;
40
41  Mbx = RqCreateMailbox(0,@Exc); call ExcChk;
42  User = CqCreateCommUser(@Exc); call ExcChk;
43
44  call ConsGet(@VCTrace,@('trace flag = 'F'B '.'));
45
46  Ref = VCOpen(Mbx,User);
47
48  call ConsGet(@TSAP,@('TSAP local = 'F'W remote = 'F'W '.'));
49  call ConsGet(@NA,@(' net = 'F'B,'F'B,'F'B,'F'B,'F'B,'F'B'↑'.'));
50
51  call ConsGet(@C,@('active/passive open ='C'↑'.'));
52  if C='a'
53  then TAP = ActTAB(TSAP(0),@NA,TSAP(1));
54  else TAP = PasTAB(TSAP(0),@NA,TSAP(1));
55  call VCConnect(Mbx,User,Ref,TAP,(C='a'));
56
```

```
$eject

C = '.';
do while C>' ';
    call ConsGet(@C,@('transmit/receive = 'C '.'));

    if C='r'
    then do;
        call ZapBytes(@Pkt,Size(Pkt));
        PktSize = VCReceive(Mbx,User,Ref,@Pkt,Size(Pkt),@EOM);
        call ConsPut(@PktSize,@('('W): '.'));
        call PutBytes(@Pkt,PktSize);
        if EOM<>0
        then call ConsPut(nil,@(' EOM'↑'.'));
        else call ConsPut(nil,@(''↑'.'));
        end;
    else if C='t'
    then do;
        call ConsGet(@PktSize,@('size = 'W'↑'.'));
        I = 0; C = 00H;
        do while I<>PktSize;
            call ConsGet(@C,@(''?'F'B '.')); Pkt(I) = C;
            I = I+1;
            end;
        call ConsGet(@EOM,@('EOM='F'B'↑'.'));
        call VCSend(Mbx,User,Ref,@Pkt,PktSize,EOM);
        end;

    end;

call VCClose(Mbx,User,Ref);

call ConsClose;
call DqExit(0);
end VCTest;
```

4.5.13. TEST/NMFTEST.SRC

```
 1  $title('NMF read/reset')
 2  $large
 3  $optimize(3)
 4
 5  NMFPoke: do;
 6
 7  $include(:me:inc/dcl.lit)
 8  $include(:sd:rmx86/inc/ltksel.lit)
 9  $include(:sd:rmx86/inc/uexit.ext)
10  $include(:sd:rmxnet/inc/cqrb.ext)
11  $include(:me:inc/fmt.ext)
12  $include(:me:inc/except.ext)
13  $include(:me:inc/nmfread.ext)
14
15  $include(:me:inc/putbyt.inc)
16
17  Echo: procedure(Rest);
18      dcl Rest pointer; /* to string */
19      call ConsPut(Rest,@(''S '.'));
20      end Echo;
```

```
21  $eject
22
23  dcl ObjFmt byte initial (0), /* format character */
24      Obj word initial (2003H), /* object number */
25      ObjModif word initial (0000H), /* subobject */
26      ObjBuf (100H) byte;
27
28  Req: procedure(Op);
29      dcl Op byte; /* operation */
30      dcl L word;
31
32      call ConsGet(@Obj,@('object='?'F'W '.'));
33      call ConsGet(@ObjModif,@('modifier='?'F'W '.'));
34      L = NMFRead(User,Op,Obj,ObjModif,Size(ObjBuf),@ObjBuf,@Exc);
35      call ExcChk;
36      if L<8
37      then call ConsPut(@L,@('error: fill length='W'↑'.'));
38      else do case (ObjFmt);
39          do; /* byte (*) */
40              call ConsPut(nil,@('('.'));
41              call PutBytes(@ObjBuf,L-7);
42              call ConsPut(nil,@(')'.'));
43              end;
44          call ConsPut(@ObjBuf,@('('F'W)'.')); /* word */
45          call ConsPut(@ObjBuf,@('('F'D)'.')); /* dword */
46          end;
47      end Req;
48
49  SetObjFmt: procedure;
50      dcl F byte;
51      call ConsGet(@F,@('='C'.'));
52      if F='b'
53      then do; call Echo(@(7,'yte (*)')); ObjFmt = 0; end;
54      else if F='w'
55      then do; call Echo(@(3,'ord')); ObjFmt = 1; end;
56      else if F='d'
57      then do; call Echo(@(4,'word')); ObjFmt = 2; end;
58      else call ConsPut(nil,@(' word, dword, or byte (*)'↑'.'));
59      end SetObjFmt;
60
61  CmdLoop: procedure;
62      dcl Ch byte; /* command character */
63      do while 1;
64          call ConsGet(@Ch,@(''↑command: 'C'.'));
65          if Ch=0DH
66          then return; /* done */
67          else if Ch='r'
68          then do; call Echo(@(3,'ead')); call Req(00H); end;
69          else if Ch='c'
70          then do; call Echo(@(4,'lear')); call Req(01H); end;
71          else if Ch='f'
72          then do; call Echo(@(5,'ormat')); call SetObjFmt; end;
73          else call ConsPut(nil,@(''↑',
74                                  'read, ',
75                                  'clear, ',
76                                  'format, ',
77                                  'or CR to stop'↑'.'));
78          end;
79      end CmdLoop;
```

```
80  $eject
81
82  dcl User word; /* comm user ID */
83
84  call ConsOpen;
85
86  User = CqCreateCommUser(@Exc); call ExcChk;
87
88  call CmdLoop;
89  call ConsPut(nil,@(''↑'.'));
90
91  call ConsClose;
92  call DqExit(0);
93
94  end NMFPoke;
```

4.5.14. UTILS/JAM.SRC

```
 1  $title('DL jamming')
 2  $large
 3  $optimize(3)
 4
 5  Jammer: do;
 6
 7  $save nolist
 8  $include(:me:inc/dcl.lit)
 9  $include(:sd:rmx86/inc/ltksel.lit)
10
11  $include(:sd:rmx86/inc/ncrseg.ext)
12  $include(:sd:rmx86/inc/ncrmbx.ext)
13  $include(:sd:rmx86/inc/nrcmes.ext)
14  $include(:sd:rmx86/inc/ndimbx.ext)
15
16  $include(:sd:rmx86/inc/ucreat.ext)
17  $include(:sd:rmx86/inc/uopen.ext)
18  $include(:sd:rmx86/inc/uclose.ext)
19  $include(:sd:rmx86/inc/uexit.ext)
20
21  $include(:sd:rmxnet/inc/cqrb.ext)
22
23  $include(:me:inc/fmt.ext)
24  $include(:me:inc/except.ext)
25  $include(:me:inc/ticker.ext)
26  $include(:me:inc/cal.ext)
27  $include(:me:inc/rbhdr.inc)
28  $include(:me:inc/dl.inc)
29  $include(:me:inc/dl.ext)
30  $include(:me:inc/zapbyt.inc)
31  $restore
```

```
32  $subtitle('')
33
34  dcl Mbx token, /* request mailbox */
35      User word; /* job ID? */
36  dcl DA (6) byte initial (255,255,255,255,255,255), /* network DA */
37      DLSAP byte; /* test AP */
38  dcl NSend dword, /* number to send */
39      NRBs word, /* degree of overlap */
40      NBytes word; /* length of packets */
41  dcl TStart dword, /* start time */
42      TSpent dword; /* elapsed time */
43  dcl PktPtr pointer,
44      Pkt based PktPtr structure
45        ( Hdr (THdrSize) byte,
46          R word, /* request block number */
47          N dword ); /* sequence number */
48  dcl R word,
49      Resp word;
50
```

```
51  $eject
52
53  /* convert absolute to pointer */
54  PtrAbs: procedure (A) pointer;
55      dcl A dword;
56      dcl P pointer,
57      Q structure (S word, O word) at (@P);
58      Q.S = Shr(A,4); Q.O = A and 1111B;
59      return P;
60      end PtrAbs;
61
62  /* make a transmit RB */
63  MakeTRB: procedure(PktPtr,PktSize) selector;
64      dcl PktPtr pointer, /* to packet */
65      PktSize word; /* of packet */
66      RBHdrSel, TRBSel = RqCreateSegment(Size(TRB),@Exc); call ExcChk;
67      RBHdr.Length = Size(TRB);
68      RBHdr.User = User;
69      RBHdr.RespPort = 0FFH;
70      RBHdr.RetMbx = Mbx;
71      RBHdr.RBSel = RBHdrSel;
72      RBHdr.SubSys = 20H;
73      RBHdr.Op = 84H;
74      RBHdr.Resp = 0;
75
76      TRB.BufCount = 1;
77      TRB.ByteCount(0) = PktSize-RHdrSize+THdrSize; /* true length */
78      TRB.BufLoc(0) = AbsPtr(@Pkt);
79      TRB.DstAddrPtr = AbsPtr(@DA);
80
81      THdrPtr = @Pkt.Hdr;
82      THdr.DstLSAP, THdr.SrcLSAP = DLSAP;
83      THdr.ISOCmd = 03H;
84
85      return RBHdrSel;
86      end MakeTRB;
87
```

```
 88  $eject
 89
 90  Jam: procedure;
 91      dcl N dword; /* packet number */
 92
 93      N = 0;
 94      do while N<NSend;
 95          RBHdrSel = RqReceiveMessage(Mbx,6000,@Resp,@Exc); call ExcChk;
 96          if Resp<>0 or RBHdr.Resp<>1
 97          then do; Exc = 0FF80H+RBHdr.Resp; call ExcChk; end;
 98          N = N+1;
 99          TRBSel = RBHdrSel;
100          PktPtr = PtrAbs(TRB.BufLoc(0));
101          Pkt.N = N;
102          call CommRB(RBHdrSel,@Exc); call ExcChk;
103          end;
104
105      end Jam;
```

```
106  $eject
107
108  call ConsOpen;
109
110  Mbx = RqCreateMailbox(0,@Exc); call ExcChk;
111  User = CqCreateCommUser(@Exc); call ExcChk;
112
113  call ConsGet(@DA,@('network DA = '?'F'B,'?'F'B,'?'F'B,'?'F'B,'?'F'B,'?'F'B '.'));
114  call ConsGet(@DLSAP,@('DLSAP = 'F'B't'.'));
115  call ConsGet(@NSend,@('packet count = 'D '.'));
116  call ConsGet(@NRBs,@('RBs = 'W '.'));
117  call ConsGet(@NBytes,@('size = 'W't'.'));
118
119  call DLConnect(Mbx,User,DLSAP);
120
121  /* stuff the pipeline */
122  R = 0;
123  do while R<NRBs;
124      PktPtr = BuildPtr(RqCreateSegment(NBytes,@Exc),0); call ExcChk;
125      call ZapBytes(@Pkt,NBytes);
126      Pkt.R = R;
127      call CommRB(MakeTRB(@Pkt,NBytes),@Exc); call ExcChk;
128      R = R+1;
129      end;
130
131  call TickStart;
132  TStart = Ticks;
133  call Jam;
134  TSpent = Ticks-TStart;
135
136  /* drain pipe */
137  R = 0;
138  do while R<NRBs;
139      RBHdrSel= RqReceiveMessage(Mbx,6000,@Resp,@Exc); call ExcChk;
140      R = R+1;
141      end;
142
143  call DLDisconnect(Mbx,User,DLSAP);
144
145  call ConsPut(@TSpent,@(''D ticks't'.'));
146
147  call ConsClose;
148  call DqExit(0);
149  end Jammer;
```

4.5.15. UTILS/DG.SRC

```
$large

Stat: do;
$include(:me:inc/dcl.lit)
$include(:sd:rmx86/inc/ltksel.lit)
$include(:sd:rmx86/inc/uexit.ext)
$include(:sd:rmxnet/inc/cqrb.ext)
$include(:me:inc/except.ext)
$include(:me:inc/fmt.ext)
$include(:me:inc/nmfread.ext)

dcl S structure
   ( ID (6) byte,
     Sent dword,
     Coll1 word,
     Coll2 word,
     Coll16 word,
     Rcvd dword,
     CRC word,
     Align word,
     Resorce word );
dcl U word,
    L word;

U = CqCreateCommUser(@Exc); call ExcChk;

L = NMFRead(U,0,2002H,0,Size(S.ID),@S.ID,@Exc); call ExcChk;
L = NMFRead(U,0,2003H,0,Size(S.Sent),@S.Sent,@Exc); call ExcChk;
L = NMFRead(U,0,2004H,0,Size(S.Coll1),@S.Coll1,@Exc); call ExcChk;
L = NMFRead(U,0,2005H,0,Size(S.Coll2),@S.Coll2,@Exc); call ExcChk;
L = NMFRead(U,0,2006H,0,Size(S.Coll16),@S.Coll16,@Exc); call ExcChk;
L = NMFRead(U,0,2007H,0,Size(S.Rcvd),@S.Rcvd,@Exc); call ExcChk;
L = NMFRead(U,0,2008H,0,Size(S.CRC),@S.CRC,@Exc); call ExcChk;
L = NMFRead(U,0,2009H,0,Size(S.Align),@S.Align,@Exc); call ExcChk;
L = NMFRead(U,0,200AH,0,Size(S.Resorce),@S.Resorce,@Exc); call ExcChk;

call ConsOpen;
call ConsPut(@S,@('ID='F'B 'F'B 'F'B 'F'B 'F'B 'F'B '↑',
                  'Sent='D primary collision='W secondary='W exceed='W'↑',
                  'Received='D CRC error='W unaligned='W overrun='W'↑',
                  ''.'));
call ConsClose;
call DqExit(0);
end Stat;
```