A Final Report
Grant No. NAG3-630

April 1, 1985 - April 14, 1989

AIRNET: A REAL-TIME COMMUNICATIONS NETWORK FOR AIRCRAFT

Submitted to:

National Aeronautics and Space Administration
Lewis Research Center
21000 Brookpark Road
Cleveland, OH 44135

Attention:

Dr. J. C. DeLaat, MS 77-1
Advanced Control Technology Branch

Submitted by:

Alfred C. Weaver
Associate Professor

Report No. UVA/528238/CS90/101
June 1990

Computer Networks Laboratory
DEPARTMENT OF COMPUTER SCIENCE

SCHOOL OF
ENGINEERING
& APPLIED SCIENCE

University of Virginia
Thornton Hall
Charlottesville, VA 22903

UNIVERSITY OF VIRGINIA
School of Engineering and Applied Science

The University of Virginia's School of Engineering and Applied Science has an undergraduate enrollment of approximately 1,500 students with a graduate enrollment of approximately 600. There are 160 faculty members, a majority of whom conduct research in addition to teaching.

Research is a vital part of the educational program and interests parallel academic specialties. These range from the classical engineering disciplines of Chemical, Civil, Electrical, and Mechanical and Aerospace to newer, more specialized fields of Applied Mechanics, Biomedical Engineering, Systems Engineering, Materials Science, Nuclear Engineering and Engineering Physics, Applied Mathematics and Computer Science. Within these disciplines there are well equipped laboratories for conducting highly specialized research. All departments offer the doctorate; Biomedical and Materials Science grant only graduate degrees. In addition, courses in the humanities are offered within the School.

The University of Virginia (which includes approximately 2,000 faculty and a total of full-time student enrollment of about 17,000), also offers professional degrees under the schools of Architecture, Law, Medicine, Nursing, Commerce, Business Administration, and Education. In addition, the College of Arts and Sciences houses departments of Mathematics, Physics, Chemistry and others relevant to the engineering research program. The School of Engineering and Applied Science is an integral part of this University community which provides opportunities for interdisciplinary work in pursuit of the basic goals of education, research, and public service.

A Final Report
Grant No. NAG3-630

April 1, 1985 - April 14, 1989

AIRNET: A REAL-TIME COMMUNICATIONS NETWORK FOR AIRCRAFT

Submitted to:

National Aeronautics and Space Administration
Lewis Research Center
21000 Brookpark Road
Cleveland, OH 44135

Attention:

Dr. J. C. DeLaat, MS 77-1
Advanced Control Technology Branch

Submitted by:

Alfred C. Weaver
Associate Professor

Report No. UVA/528238/CS90/101
June 1990

Computer Networks Laboratory
DEPARTMENT OF COMPUTER SCIENCE

A Final Report
Grant No. NAG3-630

April 1, 1985 - April 14, 1989

AIRNET: A REAL-TIME COMMUNICATIONS NETWORK FOR AIRCRAFT

Submitted to:

National Aeronautics and Space Administration
Lewis Research Center
21000 Brookpark Road
Cleveland, OH 44135

Attention:

Dr. J. C. DeLaat, MS 77-1
Advanced Control Technology Branch

Submitted by:

Alfred C. Weaver
Associate Professor

Computer Network Laboratory
Department of Computer Science
SCHOOL OF ENGINEERING AND APPLIED SCIENCE
UNIVERSITY OF VIRGINIA
CHARLOTTESVILLE, VIRGINIA

Report No. UVA/528238/CS90/101
June 1990

Copy No.____

AirNET: A REAL-TIME COMMUNICATIONS NETWORK FOR AIRCRAFT

Brendan G. Cain
Alfred C. Weaver
M. Alexander Colvin
Robert Simoncic

Computer Networks Laboratory
Thornton Hall
University of Virginia
Charlottesville, Virginia 22903

(804) 979-7529

ABSTRACT

We have developed a real-time local area network for use on aircraft and space vehicles. It uses token ring technology to provide high throughput, low latency, and high reliability. The system has been implemented on PCs and PC/ATs operating on PCbus, and on Intel 8086/186/286/386s operating on Multibus. We provide a standard IEEE 802.2 Logical Link Control interface to (optional) upper layer software; this permits the controls designer to utilize standard communications protocols (e.g., ISO, TCP/IP) if time permits, or to utilize our very fast link level protocol directly if speed is critical. Both unacknowledged datagram and reliable virtual circuit services are supported. Using our software, a station operating an 8 MHz Intel 286 as a host can generate a sustained load of 1.8 megabits per second per station, and we can deliver a 100-byte message from the transmitter's user memory to the receiver's user memory, including all operating system and network overhead, in under 4 milliseconds.

TABLE OF CONTENTS

AirNET: A REAL-TIME LOCAL AREA NETWORK FOR AIRCRAFT

1. Background

Historically, electronic communication aboard aircraft has been accomplished by using point-to-point connections or bus wiring, the latter being exemplified by the 1 Mbps, master/slave MIL-STD-1553B avionics bus. As the number and type of devices which need interconnection increase, and as the amount of data which needs to pass among them also increases, these interconnection strategies become less viable. What is needed is a high speed interconnect which can be shared by many devices — in other words, we need some form of local area network.

NASA-Lewis Research Center faced just such a problem in their attempt to accomplish distributed control of aircraft engines and airframes. The CPUs, sensors, and effectors which were physically distributed about the aircraft needed a common real-time interconnection mechanism to support their interaction. NASA-Lewis contracted with the Computer Networks Laboratory at the University of Virginia to build a prototype local area network which would support this type of real-time messaging.

Our analysis of the communications patterns aboard an aircraft suggested that our system needed to support three distinctly different types of traffic: (1) slow, periodic messages, (2) background file transfer, and (3) real-time control messages. The first two message types had negligible impact on performance, so our system was designed around the needs of the real-time control systems. Our requirements were that (1) each network station should be able to process at least one hundred 100-byte control messages per second; (2) once generated, a message should be delivered within 10 ms; (3) the network itself should support a sustained data rate of 5-10 megabits/second; and (4) each network station should be based on commercial-off-the-shelf technology, in the class of a 6-12 MHz Intel 286 or 386 microprocessor.

We acquired a commercial token ring network, the Proteon ProNET-10, and developed around it a user-friendly yet robust real-time messaging system. This was equivalent to providing all the

communications services within the kernel of a modern operating system. We call the result AirNET.

2. *User Interface*

One possibility for the user interface would be the full suite of ISO protocols; such a choice would have provided a messaging service in the application layer such as X.400 or the Manufacturing Message Specification (MMS). But we knew from our previous network performance evaluations [2,3,4] that the commercially available ISO protocol packages (1) would not meet our performance requirements and (2) were really overkill for a relatively short, single-segment LAN with a modest number of stations. Therefore we chose to write our own user interface, provided as a set of library functions which the user links into his application program. To encourage interoperability, our system adheres strictly to the IEEE 802.2 Logical Link Control (LLC) standard.

AirNET provides a basic datagram service, with optional acknowledgements and checksums, to multiple application processes running on microcomputers. The user interface is a set of 'C' procedure calls which create and manage sockets, set options, send and receive messages, and report network status.

The LLC architecture is shown in Figure 1. Communication occurs through *sockets*, which are equivalent to IEEE 802.2 LSAPs (link service access points). Programmers use the LLC interface by linking into their 'C' programs as many of the following communications primitives as are needed for the intended application:

LLCon (socks) initializes the network interface. Table space for *socks* number of sockets is allocated.

LLCoff( ) disables and closes the socket interface.

LLCopen (sock) opens the socket numbered *sock*. A socket must be opened before use. Like LSAPs, sockets must be even-numbered integers in the range 2 to 254 inclusive.

LLCclose (sock) closes the socket numbered *sock*. The socket must be idle (no pending transmit or

receive calls).

LLCoption (sock, xsignal, rsignal, xtime, rtime, priority, ack, tries) sets options on socket number *sock*. Variables *xsignal* and *rsignal* are pointers to functions to be called when a packet has been transmitted or received, respectively. The variables *xtime* and *rtime* are the amount of time allowed for the operation to complete (a value of zero never times out). The value of *priority* sets the transmission priority of the message in the range 0 (lowest) to 7 (highest). The LLC software manages an eight-level queue, serving the highest priority messages first. The flag *ack*, if set, requires that the packet be specifically acknowledged by a reply message. If the transmission was not successful, *tries* tells how many times the transmitter should transparently send the packet before declaring an error.

LLCxmit (sock, destination, packet, size) delivers the packet pointed to by *packet*, of length *size*, to socket number *sock* for delivery to the network entity whose address is pointed to by *destination*.

LLCrecv (sock, source, packet, size) enables reception of a packet at socket *sock*. The programmer provides *source* (a pointer to a buffer to hold the incoming packet's source address), *packet* (a pointer to a buffer that will hold the incoming packet), and *size* (the length of the packet buffer).

LLCxmit and **LLCrecv** move messages between the LLC entity and the MAC (medium access control) engine in the same computer (not end-to-end). This frees the CPU to operate in parallel with the network hardware. Using IEEE 802.2 terminology, the procedure call represents the data *request*, the return from the call represents the *confirm*, and an appropriate change in the status byte returned by the **LLCstatus** call represents the *indication*.

LLCreset (sock, direction) resets any pending operation on the transmit side (if *direction* = *'x'*) or receive side (if *direction* = *'r'*) of the socket and releases the associated buffer. A socket is bidirectional and so can send and receive simultaneously.

LLCstatus (sock, direction, status) returns the current status of a socket. For socket *sock*, on the

transmit or receive side as determined by *direction*, **LLCstatus** returns *status*, a pointer to a status variable, which indicates operation pending, no operation pending, I/O in progress, operation failed, or operation timed out.

Finally, every operation on a socket returns an operation status: operation accepted, invalid socket, duplicate socket, too many sockets, socket busy, or packet too large.

We also provide a real-time network monitor which displays color-coded histograms of recent network traffic. The monitor can trace all network traffic, or selected traffic as defined by a user-specified filter. The monitor displays network load in packets/sec and in bits/sec, sampled at a user-defined rate, and calculates current, average, and maximum data rates. In trace mode the monitor can trap and later display the last 1,000 network events.

3. Example Programs

To show the simplicity of the user interface, we present two example programs. The first is extremely simple: a transmitter broadcasts an unacknowledged datagram on the network. The second is more complex: the sender creates a reliable virtual circuit to guarantee in-order delivery with no duplicates. This requires the use of higher-level acknowledgements and sequence numbers.

Services are provided by a software library containing the aforementioned LLC procedure calls. When the LLC service is used, those LLC calls must be issued in a certain sequence to obtain the desired result. The network interface must first be initialized, then one or more sockets may be opened for communication and their options set. Only then can receive and transmit operations be applied. Each socket can be used for both reception and transmission in parallel; receive and transmit operations are full duplex. To close a connection the socket should first be reset, then closed. To stop all communication, each socket should be reset and closed, then the communications disabled.

If communication is to be established between two machines, one a receiver and the other a transmitter, a possible sequence of operations could be as follows.

At the transmitter:

1. **LLCon** will initialize the network interface and allocate table space for the socket.
2. **LLCopen** will open the socket.
3. **LLCoption** will set options on the socket for transmission.
4. Several **LLCxmit** operations can be issued to send a number of packets through the socket to another socket on the remote machine.
5. After all information is transmitted, **LLCreset** will reset the state of the socket.
6. **LLCclose** will close the socket.
7. **LLCoff** will disable and deallocate the socket interface.

At the receiver:

1. **LLCon** will initialize the network interface and allocate table space for the socket.
2. **LLCopen** will open the socket.
3. **LLCoption** will set options on the socket for reception.
4. The appropriate number of **LLCrecv** operations should be issued to receive all packets from the remote machine.
5. After all information has been received, **LLCreset** will reset the state of the socket.
6. **LLCclose** will close the socket.
7. **LLCoff** will disable and deallocate the socket interface.

3.1. Example 1 - A Broadcast Datagram Service

To illustrate LLC operations we offer two examples. Both are written in Turbo C 1.5. The first program, **blast**, can be run on any station. It broadcasts MSND messages of fixed length SIZE to all stations on the network.

```
/*********************************************************/
/*   Program "blast" will broadcast MSND messages        */
/*   of size SIZE to all nodes in the network.           */
/*********************************************************/

#include       <stdio.h>              /* standard I/O  */
#include       <llcio.h>              /* LLC interface */
#define        MSND    100            /* number of messages to send */
#define        SIZE    1000           /* size of the message     */
#define        BROADC  255            /* broadcast address */

int     cls     =       0,            /* service class */
        tout    =       0,            /* timeout       */
        retry   =       1,            /* retry limit   */
        ns      =       2,            /* source socket */
        mnum;
unsigned
  int  status;                        /* returned status */
char    message[SIZE]   =       { "Message..."};   /* broadcast message */
unsigned
  char netadr[2]  =  { 2, 255 };      /* destination: socket number, node address */

nop() { }                             /* signal function */

down(opn)                             /* close down */
  char opn;
  { switch (opn) {
    case 'r' :
        while  (status = LLCreset(ns,'x'))
            printf("*** LLCreset rejected, status : %s\n ", LLCopbits(status));
    case 'c' :
        if (status = LLCclose(ns))
            printf("*** LLCclose rejected, status : %s\n", LLCopbits(status));
    case 'o' :
        if  (status = LLCoff())
            printf("*** LLCoff rejected, status : %s\n", LLCopbits(status));
    default:
        exit(1);
   }
}

main()                                /* transmit MSND messages */
  {
   /* initialize the network, open the socket and set the options */

   if  (status = LLCon(1)) {
     printf("*** LLCon rejected, status : %s\n", LLCopbits(status));
     exit(1);
   }
   if  (status = LLCopen(ns)) {
     printf("*** LLCopen rejected, status : %s\n", LLCopbits(status));
     down('o');
   }
   if (status = LLCoption(ns, nop, nop, tout, tout, cls, retry)) {
     printf("*** LLCoption rejected, status : %s\n", LLCopbits(status));
     down('c');
   }

   /* transmit MSND messages, wait for non-busy status after each transmission */

   for (mnum = 1; mnum < MSND; mnum++) {
     if (status = LLCxmit(ns, netadr, message, sizeof(message))){
       printf("*** LLCxmit rejected, status : %s\n", LLCopbits(status));
       down('r');
     }
     while (LLCstatus(ns, 'x', &status), (status&STBUSY))
               ; /* wait until not busy */
   }
   down ('r'); /* close down */
 }
```

Program **blast** starts by initializing the network interface and allocating table space for one socket by calling **LLCon**. If **LLCon** is successful, **LLCopen** is called to open the socket.

LLCoption is used to set the socket options. If the returned status indicates unsuccessful operation, an error message is printed and the program exits with **down('c')** which will call **LLCclose** and **LLCoff** before exiting. A successful call to **LLCoption** in this program will set the following options for the socket ns=2:

1. The signal routine for transmit and receive is the same — **nop()** which does nothing.
2. Timeout for transmit and receive is set to 0, which means that operation will never be timed out.
3. Class is set to 0 — connectionless data link service without acknowledgements. Priority is 0 (lowest).
4. Number of transmission attempts is 1; LLC will not try to retransmit the packet if the first attempt fails.

Now the program calls **LLCxmit**. After each transmission, the return status of the operation is tested; if it indicates an error, the program exits. Otherwise it waits for the non-busy status of the socket, indicating that the socket is ready, and then issues the next transmit operation.

3.2. Example 2 - Reliable Virtual Circuit Service

In the second example we present one way to build a reliable virtual circuit service using the basic datagram service. It consists of two programs running on different stations. Program **sendrel** will send messages from a certain socket to the remote machine where program **recvrel** receives them. Then it waits for an acknowledgement from the receiver at the same socket. If the acknowledgement does not arrive within the specified timeout, the sender will retransmit the message and again wait for an acknowledgement. The sender will tag the messages with one bit sequence numbers, 0 or 1. For every new message transmitted, the receiver will alternate the sequence number. When retransmitting a lost

message, of course, the sequence number stays the same.

Program **recvrel** will receive messages from the sender. On each reception it will test the sequence number. If it is the same as the previous one, it means the message is being retransmitted by the sender so the receiver just emits another acknowledgement. If the sequence number is different from the previous one, the receiver will process the message, send the acknowledgement, and await another message.

In this example the communication service is reliable. If either the message or the acknowledgement is lost it will cause retransmission from the side of the sender. The receiver correctly handles duplicate messages. The source code of **sendrel**:

```
        /*********************************************************************/
        /* "sendrel" - transmitter for the reliable communication service  */
        /*********************************************************************/

#include        <stdio.h>
#include        <llcio.h>               /* LLC interface */
#define         MSND    100             /* number of messages to send */

int     cls     =       0,              /* service class */
        toutr   =       60,             /* timeout units */
        touts   =       0,
        retry   =       0,              /* retry limit   */
        ns      =       2,              /* source socket number */
        dones   =       0,               /* set when message sent */
        doner   =       0,              /* set when acknowledgement received */
        rpt     =       10,             /* retries */
        mnum;
struct  {
        unsigned char seqn;             /* sequence number */
        char    message[100];           /* message sent    */
        } dout;
unsigned
int     status1,                        /* status returned */
        status2;
unsigned
char    netadr[2]  =  { 2, 100 };       /* destination: socket number, node address */
char    seqack;                         /* received acknowledgement */

/* receive signal handler  */
sigr() {if (LLCstatus(ns, 'r', &status1), !(status1&STBUSY)) doner++;}

/* transmit signal handler */
sigs() {if (LLCstatus(ns, 'x', &status2), !(status2&STBUSY)) dones++;}

opstat(s,tag)                   /* check the operation status and return it */
  int s;
  char  *tag;
  { if (s)
      printf("*** LLC%s rejected, status: \n \t %s \n", tag, LLCopbits(s));
      return (s);
  }

down(opn)                       /* close down */
  char opn;
  { switch (opn) {
```

```
      case 'r' :  while  (opstat(LLCreset(ns,'r'), "reset") ) ;
      case 'c' :  opstat(LLCclose(ns), "open");
      case 'o' :  opstat(LLCoff(), "off");
      default  :  exit(1);
    }
  }

main()                          /* reliable transmitter */
  {
    /* initialize, open and set options for the socket */
    if ( opstat(LLCon(1), "on") )  exit(1);
    if ( opstat(LLCopen(ns), "open") )  down ('o');
    if ( opstat(LLCoption(ns,sigs,sigr,touts,toutr,cls,retry), "option") ) down ('c');

    sprintf(dout.message, "MESSAGE....");     /* initialize out message */
    dout.seqn = 0;                            /* initialize seq number  */

    /* send MSND messages with 10 retries upon failure */

    for (mnum = 1; mnum < MSND; mnum++) {
        if ( rpt == 0 )  down('r');           /* if 10 retries -> down! */
        /***.........PREPARE THE MESSAGE HERE.............***/
        dout.seqn = (dout.seqn == 0) ? 1 : 0 ;        /* flip sequence number */
        printf( " Sending %d %s \n", dout.seqn, dout.message );
        rpt  =  10;     /* transmission retries */
        do {
            doner = dones = 0;         /* signal handler indicators */
            /* receive acknowledgement  */
            if(opstat(LLCrecv(ns, netadr, &seqack, sizeof(seqack)), "recv"))  down('r');
            /*  send message */
            if(opstat(LLCxmit(ns, netadr, &dout, sizeof(dout)), "xmit")) down('r');
            while ( !dones  || !doner  );     /* wait until done  */
            if (status1&STFAIL) {             /* transmission failed */
                printf("*** hexstat = %x\n", status1);
                printf("*** transmit failed, status: %s\n", LLCstbits(status1,'x'));
            }
            else
                if ( dout.seqn  == seqack )
                        break;                /* O.K. -> send next message */
                   else
                        down ('r' );          /* error -> down */
        } while ( (rpt--) , ( rpt > 0 ));
    }
     down('r');
}
```

Program **sendrel** makes use of signal handlers for both transmit and receive operations. First **LLCstatus** is called to get the transmit or receive status of the socket; if it is not busy **doner** or **dones** is incremented to indicate the completion of the operation.

The main module first initializes the network interface, opens the socket, sets options for the socket and initializes the output message. Then it starts transmission and waits for the acknowledgement. If the acknowledgement does not arrive within the timeout the message is retransmitted up to 10 times. If the sequence number is out of order the program exits, else the next message is transmitted.

Program **recvrel** is the reliable receiver for **sendrel**. It runs on another station.

```
/*****************************************************************/
/* "recvrel" - receiver for the reliable communication service   */
/*****************************************************************/

#include      <stdio.h>
#include      <llcio.h>              /* LLC interface */
#define       MRCV    1000           /* number of messages to receive */

int     cls     =       0,           /* service class */
        toutr   =       100,         /* timeout units */
        touts   =       0,
        retry   =       0,           /* retry limit   */
        ns      =       2,           /* source socket number */
        dones   =       0,            /* set when acknowledgement sent */
        doner   =       0,           /* set when message received */
        mnum;
struct  {
        unsigned char seqn;          /* sequence number  */
        char    message[100];        /* message received */
        } din;
unsigned
int     status1,                     /* status returned */
        status2;
unsigned
char    netadr[2]  =  { 2, 5 };       /* remote: socket number, node address */
char    seqack =      0;             /* acknowledgement - sequence number */

/* receive signal handler  */
sigr() {if (LLCstatus(ns, 'r', &status1), !(status1&STBUSY)) doner++;}

/* transmit signal handler */
sigs() {if (LLCstatus(ns, 'x', &status2), !(status2&STBUSY)) dones++;}

opstat(s,tag)                  /* check the operation status and return it */
  int s;
  char  *tag;
  { if (s)
      printf("*** LLC%s rejected, status: \n \t %s \n", tag, LLCopbits(s));
      return (s);
  }

down(opn)                      /* close down */
  char opn;
  { switch (opn) {
      case 'r' :  while  (opstat(LLCreset(ns,'r'), "reset") ) ;
      case 'c' :  opstat(LLCclose(ns), "open");
      case 'o' :  opstat(LLCoff(), "off");
      default  :  exit(1);
    }
  }

main()                         /* receiver for reliable communication */
  {
    /* initialize, open and set options for the socket */
    if ( opstat(LLCon(1), "on") )  exit(1);
    if ( opstat(LLCopen(ns), "open") )  down ('o');
    if ( opstat(LLCoption(ns,sigs,sigr,touts,toutr,cls,retry), "option") ) down ('c');
    /* post the initial receive buffer for the message to arrive */
    if(opstat(LLCrecv(ns, netadr, &din, sizeof(din)),"recv"))  down('r');

    /* receiving messages and sending acknowledgements */

    for (mnum = 1; mnum <= MRCV; mnum++) {
        while ( !doner  ) ;            /* wait until receive done   */
        dones = doner = 0;             /* signal handler indicators */
```

```
        /* post another receive buffer for the message  */
        if(opstat(LLCrecv(ns, netadr, &din, sizeof(din)),"recv"))  down('r');

        /* receive failed or timeout? */
        if (status1&(STFAIL|STLATE)) {
           printf("*** hexstat = %x\n", status1);
           printf("*** transmit failed, status: %s\n", LLCstbits(status1,'x'));
           down('r');
        }
        /* is this the next message ? */
        if ( din.seqn != seqack ) {
           seqack = din.seqn;                  /* set the acknowledgement seq.number */
           /* send the acknowledgement */
           if(opstat(LLCxmit(ns, netadr, &seqack, sizeof(seqack)), "xmit")) down('r');
           while (!dones) ;     /* wait until done  */
           /***........PROCESS THE MESSAGE .........***/
           printf(" received: %d %s \n", din.seqn, din.message);
        }
        else {                                 /* retransmitted message */
           printf(" the same message \n");
           /* send the old acknowledgement */
           if(opstat(LLCxmit(ns, netadr, &seqack, sizeof(seqack)), "xmit")) down('r');
           while (!dones) ;     /* wait until done  */
        }
     }
  }
```

After initializing the network, opening a socket, and setting its options, **recvrel** starts receiving messages from **sendrel** and acknowledges them one at a time by sending back the sequence number that was received in the message from the transmitter. If the receive operation fails or times out, the program exits.

Programs **sendrel** and **recvrel** are written to run in parallel on two different stations. Together they guarantee reliable exchange of information.

4. Our System

To determine whether this architecture and implementation would satisfy NASA's requirements, we built a prototype system. The resulting system consists of several workstations connected to a commercial token ring, the Proteon ProNET-10. The workstations are of three different architectures — it is the station's effect on the overall performance of the LAN that is the emphasis of this paper.

4.1. Proteon ProNET-10

The ProNET-10 is a 10 Mbit/sec token ring local area network whose functional characteristics match well with those required by our prototype system. It provides a star-ring architecture using wiring centers to interconnect up to 255 nodes or stations. Once a station has accessed the token and thus obtained the right to transmit, it may transmit a single packet of maximum length 2046 bytes, followed immediately by a new token. Placing the new token immediately after the transmitted packet, as opposed to transmitting fill until the packet has made a complete circuit, results in improved throughput in a system using primarily short messages.

4.2. Stations

The three machine architectures are:

- A Leading Edge model 'D' — an IBM PC compatible with a 7.16 MHz 8088 CPU. In this paper, this architecture is referred to by **PC**.

- A Compaq Portable — an 8 MHz 80286 — we use the term **PC/AT**.

- An INTEL 310 — a 6 MHz 80286 with a Multibus I interface — the term **INTEL** refers to this architecture.

The ProNET-10 interface for the PC and PC/AT is a single board set plugged directly into the backplane. For the INTEL, the ProNET-10 interface is a two-board set connected to the Multibus I. Both sets contain the transmit and receive hardware packet buffers and the control/status registers — effectively the MAC engine.

The LLC layer and measurement programs were implemented in the C language on the PC and PC/AT, and compiled under both the large and small models using two compilers, C86 and Turbo-C. We will see the effects of these two factors reflected in our measurements. The INTEL version was implemented under the RMX86 operating system using primarily C, although the operating system's

native language, PL/M (a subset of PL/I), was required for some of the interrupt handling and I/O. The interrupt handling was implemented using the RMX86 task system. Although the task system provides a convenient abstraction for interrupt handling, the overhead associated with resuming a task after an interrupt is significant, as we will see later. The C portion was compiled under the large model using the iC86 compiler.

Although NASA is interested primarily in the performance of a system containing only INTEL 310 stations, the inclusion of the PC and PC/AT allows us to make some interesting observations.

5. Performance and Delays

We now come to the focus of our paper — performance and delays in a real-time messaging system.

The performance of the ProNET-10 at the PHY (physical) and MAC layers is interesting in its own right, and similarly, performance of a network under factors such as high offered load and many stations is of interest when analyzing a particular protocol. However, this paper addresses primarily *station performance* at the LLC layer — the first layer whose performance is directly observable by the user. In our prototype system we find that the overall performance of the system is a direct function of an individual station's ability to transmit and receive packets at the LLC layer. The performance of the station as dictated by such factors as its DMA channel design, processor speed, and interrupt handling mechanisms, determines the system performance.

To see why this is so, let's look at the classic components of the overall end-to-end delay of a packet at the LLC layer. We define end-to-end delay as the elapsed time from the moment a message is enqueued by the sending application process until the moment the message is delivered to the receiving application process (i.e., user memory to user memory).

The hardware packet buffer holds a single packet, and so *queueing delay* at the PHY layer is negligible. Queuing delay is insignificant at higher layers due to low station offered load. Since our delay measurements were taken with a single station transmitting, *network access delay* (waiting for the

token) is also insignificant. In our prototype system, the physical ring consists solely of a wiring center (and in the typical system, ring length is short), and thus *propagation delay* is negligible. *Transmission delay* is just the packet size divided by the transmission rate of 10 Mbit/sec, and is about 1.64 milliseconds for the maximum packet size of 2046 bytes, and only 80 microseconds for a 100-byte message.

The remaining component, and the dominant factor in the end-to-end delay of a message, is *station delay*. Station delay can be decomposed into three parts on both the transmit and receive ends. A CPU-bound portion during which the packet header is filled or analyzed, followed by a DMA-bound portion in which the packet header and body are transferred (often separately) to or from the hardware packet buffer. The third and perhaps surprising portion is the interrupt handling and context switch overhead required when the DMA is complete. Delays in these three areas overwhelm the other delays, and our performance measurements illustrate the relative effects of three different machine architectures on each portion.

5.1. Station Transmit Metrics

We present the metrics for station transmission over a range of packet sizes. All sizes are in terms of information bytes only — framing bits at the LLC and lower layers are not included.

Figure 2 shows end-to-end delays for the PC, PC/AT, and INTEL machines. End-to-end delay measurements are important in real-time control systems, where bounded message delivery times are critical.

These measurements were obtained using the same station as both the transmitter and receiver, i.e., sending the message to one's self. This approach eliminates the synchronization that otherwise would be necessary to measure delays for messages sent between two stations.

We see that the INTEL performance is superior (lower delays are incurred) for packet sizes greater than 8 bytes, and that the PC/AT outperforms the PC except for large packet sizes. Note that the station

was doing no other work, and thus these numbers represent a lower bound on delay.

The delays for the PC are uniformly higher than the INTEL's because the CPU-intensive portion (accessing the socket, and filling or analyzing the packet header) is slower due to differences in the CPU. Also, setting up the DMA is slower, although the DMA itself (the per-byte transfer rates) are equivalent.

The PC/AT is much faster than the PC for shorter messages because the CPU is faster. However, the PC/AT's DMA transfer rate is slower than either the PC or INTEL, and so for large messages, begins to significantly influence end-to-end delays.

Figures 3 and 4 show the performance curves for the PC, PC/AT, and INTEL machines in packets per second and kilobits per second, respectively, over the range of packet sizes. The packets/second metric is useful if one is interested in the message rate, while the kilobits/second metric represents the maximum offered load of a station when sending messages of a certain size. These measurements were obtained using a program that continually requests a message transmission after completion of the previous request — again, the station is doing no other work — and thus these measurements represent the upper limits on station-generated load.

For these metrics, we again see that the INTEL performance is superior for packet sizes greater than about 8 bytes. The PC/AT performance exceeds the PC up through packet sizes of about 1000 bytes.

The justification for our end-to-end delay measurements also suits the packets/second and kilobits/second curves, although the end-to-end delay measurements are more dramatic since the station's effect is doubled.

5.2. Other Effects

Figures 2, 3, and 4 encapsulate the essence of the station performance we wish to present here, but there are several other interesting observations worthy of note.

5.2.1. Interrupt Handling Overhead

In our LLC architecture (see Figure 1), a packet is transmitted from the application to the hardware buffer in three stages. First, the header is moved via DMA out of the transmit engine, and second, the data itself is moved via DMA into the buffer immediately trailing the header. Third, the packet is "originated" — the MAC engine is instructed to access the next token and place the contents of the packet buffer onto the ring.

Early in the evolution of the system on the INTEL machine, the LLC transmit engine was notified of the completion of each stage through an interrupt. That is, the MAC engine interrupted the LLC layer when the request was completed. Under the RMX86 task system, the interrupt resulted in the resumption of a waiting task, which then called the appropriate routine in the transmit engine.

The performance of the INTEL system with this LLC architecture was no better than that of the PC. We then changed the architecture so that rather than being interrupted after the completion of each of the first two stages, the transmit engine just performed busy-waiting — i.e., detecting completion by continually polling the appropriate MAC control/status register. This single change resulted in the performance improvement we saw in Figures 2-4.

Thus, the overhead in our use of the task system for interrupt handling is significant. We have measured the overhead to be about 1 millisecond. This is an example of the (unexpected) cost of various operating services. Since the current architecture still requires one interrupt on packet transmission (and another on packet reception), this single factor alone contributes 2 milliseconds to the end-to-end delay of a packet, regardless of its size.

5.2.2. CPU Idle Time

A related measurement is the percentage of CPU idle time during packet transmission. We monitor the amount of time the CPU is idle while running the programs we use to determine packets/second and kilobits/second. This is a measure of the amount of time the application will be able to do other useful

work, and is a function of the packet size. In addition to the machine cycles available during DMA, the time the MAC engine spends placing the packet on the ring (the transmission delay) is time the CPU can use for alternate work.

On the INTEL, there is virtually no free time — even for long packets the free time is less than 1 percent. While this metric is initially disconcerting, the fact that packet transmission is CPU-bound leads us to believe that use of a faster CPU (e.g., the new 16-25 MHz 80386) will directly improve the station's performance.

5.2.3. Measurement Overhead

The instrumentation of a system for measurement purposes often contributes some overhead to the system, and skews the resulting measurements. We have measured that overhead, and determined it to be insignificant, on the order of 1 to 5 percent, as shown in Figure 5.

5.2.4. Same Station Effect

As mentioned earlier, end-to-end delay was measured using the same station as the transmitter and receiver — a message is sent and received, the time is recorded, another is sent and received, recorded, and so on. The resulting metric is accurate since the transmit engine and receive engine are not competing simultaneously for station resources. However, for the packets/second and kilobits/second measurements, we used a receiver separate from the transmitter. If we use the same station for both transmitting and receiving, we see a decrease in performance ranging from 10 to 30 percent over the packet size, also shown in Figure 5.

Since the receiver is handling one message while the transmitter is generating another, the receive engine competes with the transmit engine for station resources, thus slowing the overall message generation rate.

5.2.5. A Faster Transmitter

The performance curves in Figures 3 and 4 were obtained using a program constructed as follows.

The test program (the application) would request a packet transmission [via LLCxmit( )], having earlier indicated [via LLCoption( )] the application routine to call upon transmit completion. This transmit interrupt complete routine would return, and the application would then request another transmission. To save the overhead of that context switch, a load program was constructed that, rather than returning to the main application program, would reschedule another transmission from within the transmit complete interrupt routine. This approach resulted in a decrease of nearly 1 millisecond per packet transmission. This confirms our earlier measurement for interrupt handling overhead. The conclusion is that it is possible to construct a faster transmitter than those producing the performance shown in Figures 3 and 4, although any production system would probably not use such a construction.

5.2.6. Compilers

Only one compiler (iC86) was available on the INTEL, and only the large model was used because of complications with compiling our interrupt handling system under the small model. However, the PC and PC/AT had two compilers (C86 and Turbo-C) available, and the test programs were compiled under both the small and large models using each compiler. The performance of the various resulting versions were compared, and we noted significant differences. Figure 6 shows the packets/second performance curve, with the INTEL curve included for reference. The kilobits/second and end-to-end delay curves show a similar effect and thus are not included.

We note a difference in the model, large and small, with the small model providing consistently better performance. We noted a similarly consistent improvement of the performance of the Turbo-C version over that of the C86 version. On the PC, the Turbo-C large model and the C86 small model were roughly equivalent.

Note also that in Figures 2-4 shown earlier, the PC and PC/AT curves are based on the C86 large model version. We believe that version is most appropriate for comparison with the INTEL iC86 large model.

6. Conclusions

We have presented the architecture and performance of a real-time messaging system. The system is constructed using the Proteon ProNET-10 for the PHY and MAC layers, and a set of C routines comprising the LLC layer, conforming to IEEE 802.2. The system is tailored for applications characterized by few stations, predominantly short messages, and requiring bounded message delivery times. We claim the overall performance of such a system is dictated by the station performance rather than the underlying network.

We show the often surprising effect of such factors as DMA channel design, interrupt-handling, and compiler efficiency. Operating system services provide a convenient abstraction, but at a real and measurable cost.

The cumulative effect of these factors easily dominates the effects of other aspects of the network. Thus, performance improvement efforts should be focused on the station.

7. Acknowledgements

The authors gratefully acknowledge the financial support of NASA-Lewis Research Center, and the technical support of Mr. John DeLaat at NASA-Lewis.

REFERENCES

[1] PROTEON *ProNET-10 Model p1200 Multibus Local Network System Operations and Maintenance Manual*, version 4.0, part number 188-031.

[2] W. Timothy Strayer and Alfred C. Weaver, "Performance Measurements of Data Transfer Services in

MAP," *IEEE Networks*, May 1988.

[3] W. Timothy Strayer and Alfred C. Weaver, "Performance of Motorola's Implementation of MAP," *13th Local Computer Networks Conference*, Minneapolis, MN, October 1988.

[4] Jeffery H. Peden and Alfred C. Weaver, "Are Priorities Useful in an 802.5 Token Ring?", *IEEE Transactions on Industrial Electronics*, Vol. IE-35, No. 3, August 1988.

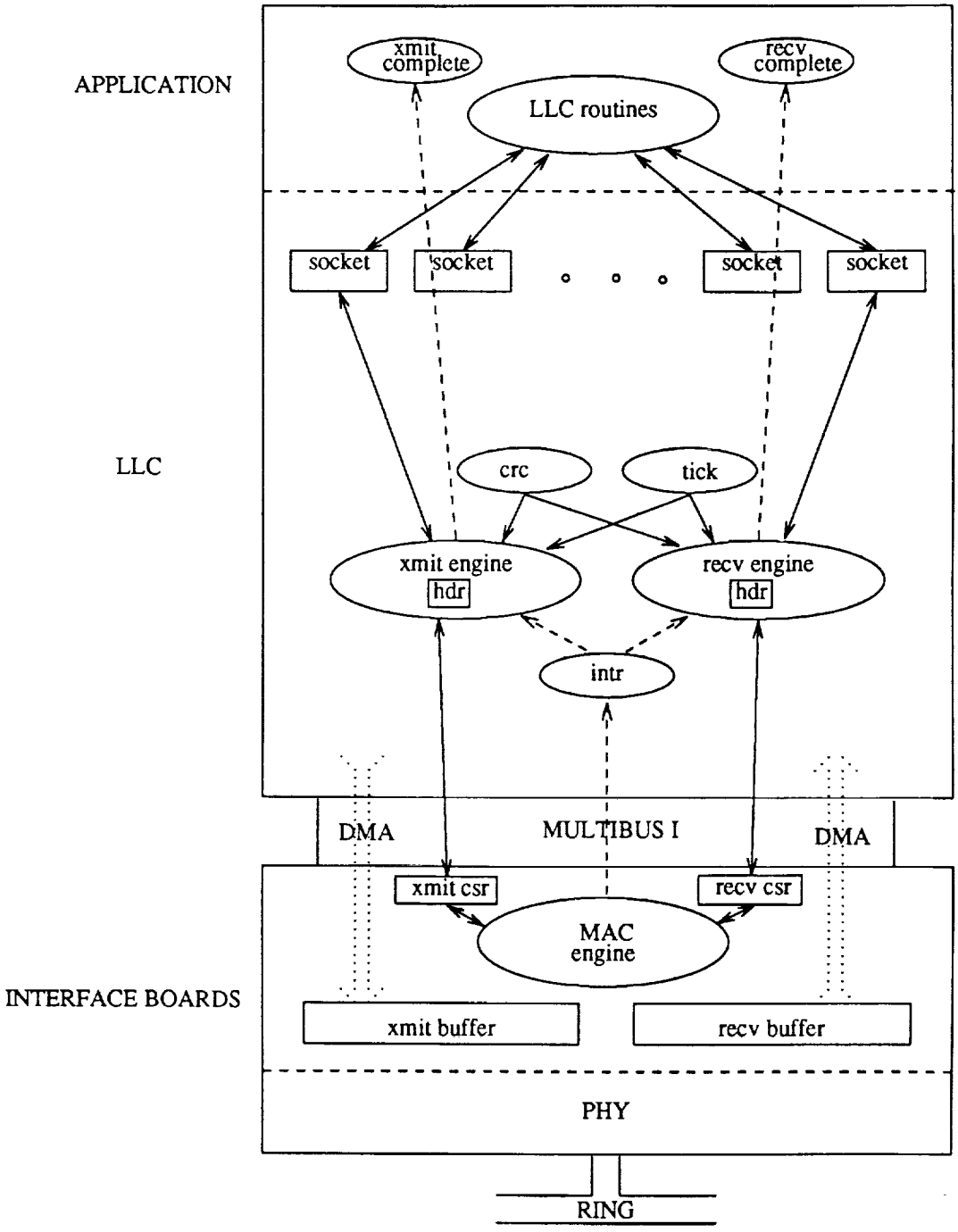


Figure 1
LLC Architecture

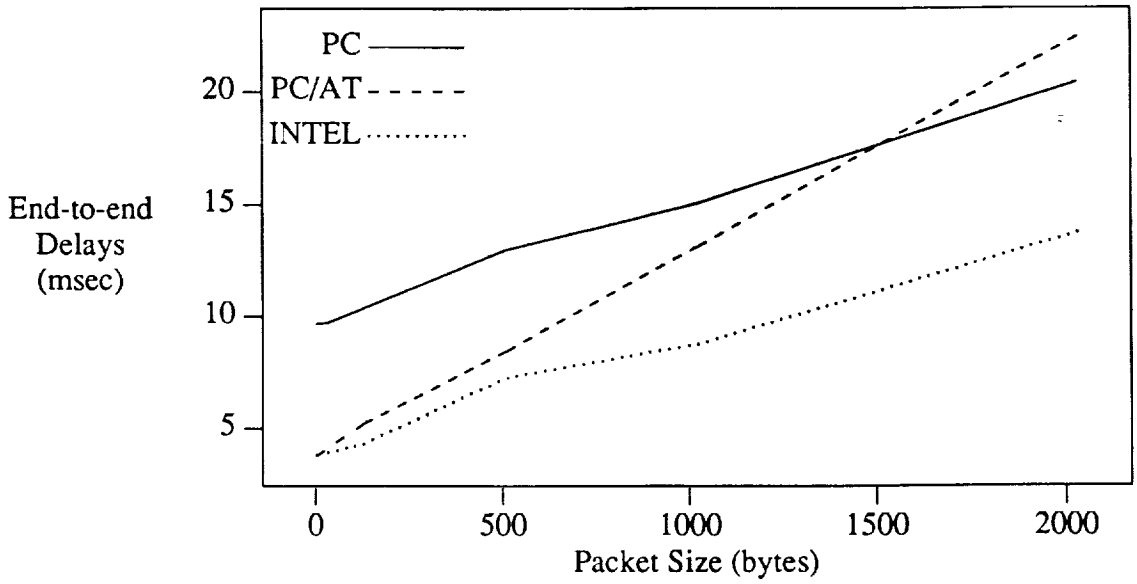


Figure 2
End-to-end Delays (msec) vs. Packet Size (bytes)

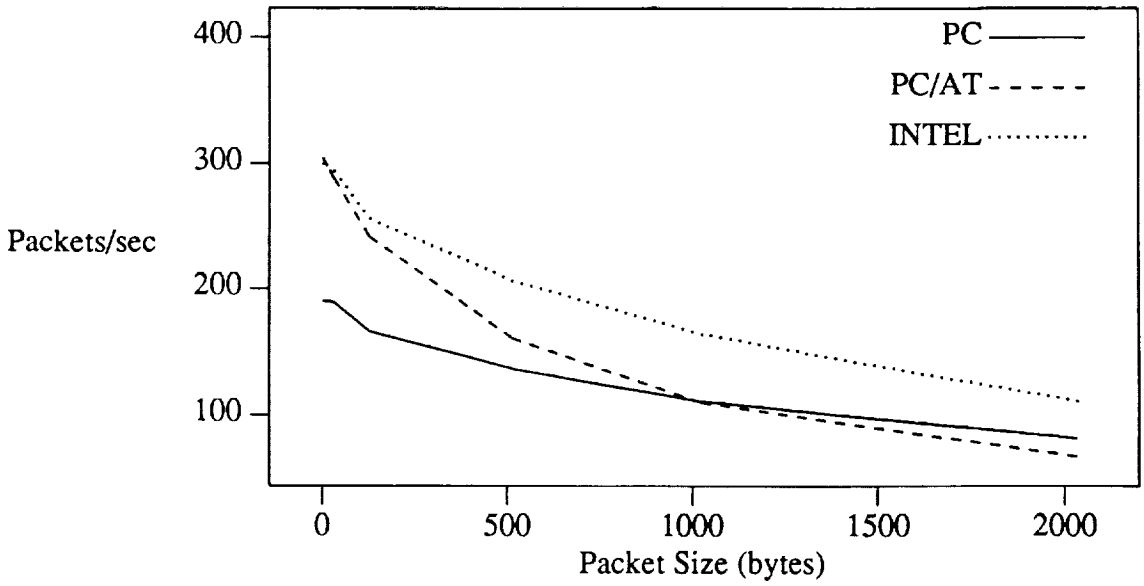


Figure 3
Transmission Rate (packets/sec) vs. Packet Size (bytes)

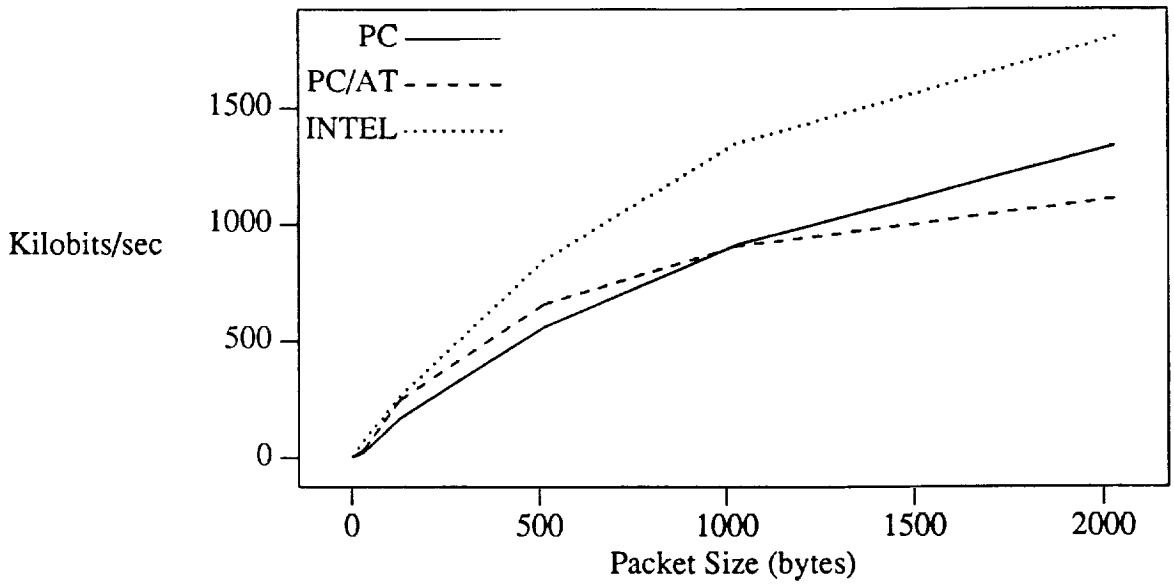


Figure 4
Transmission Rate (kilobits/sec) vs. Packet Size (bytes)

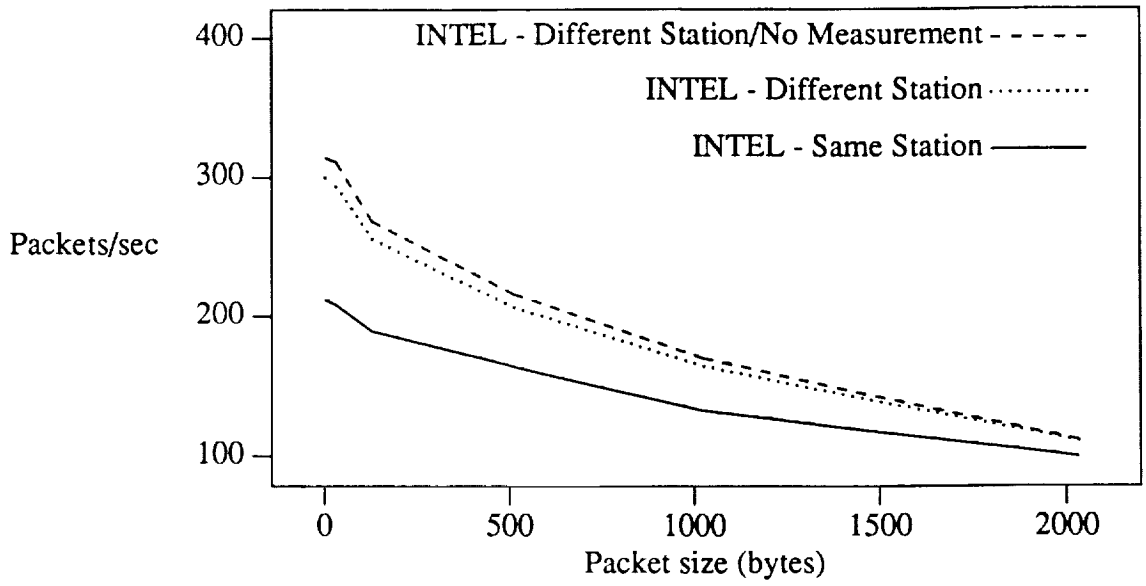


Figure 5
Transmission Rate (packets/sec) vs. Packet Size (bytes)
Showing "Same Station" Effect

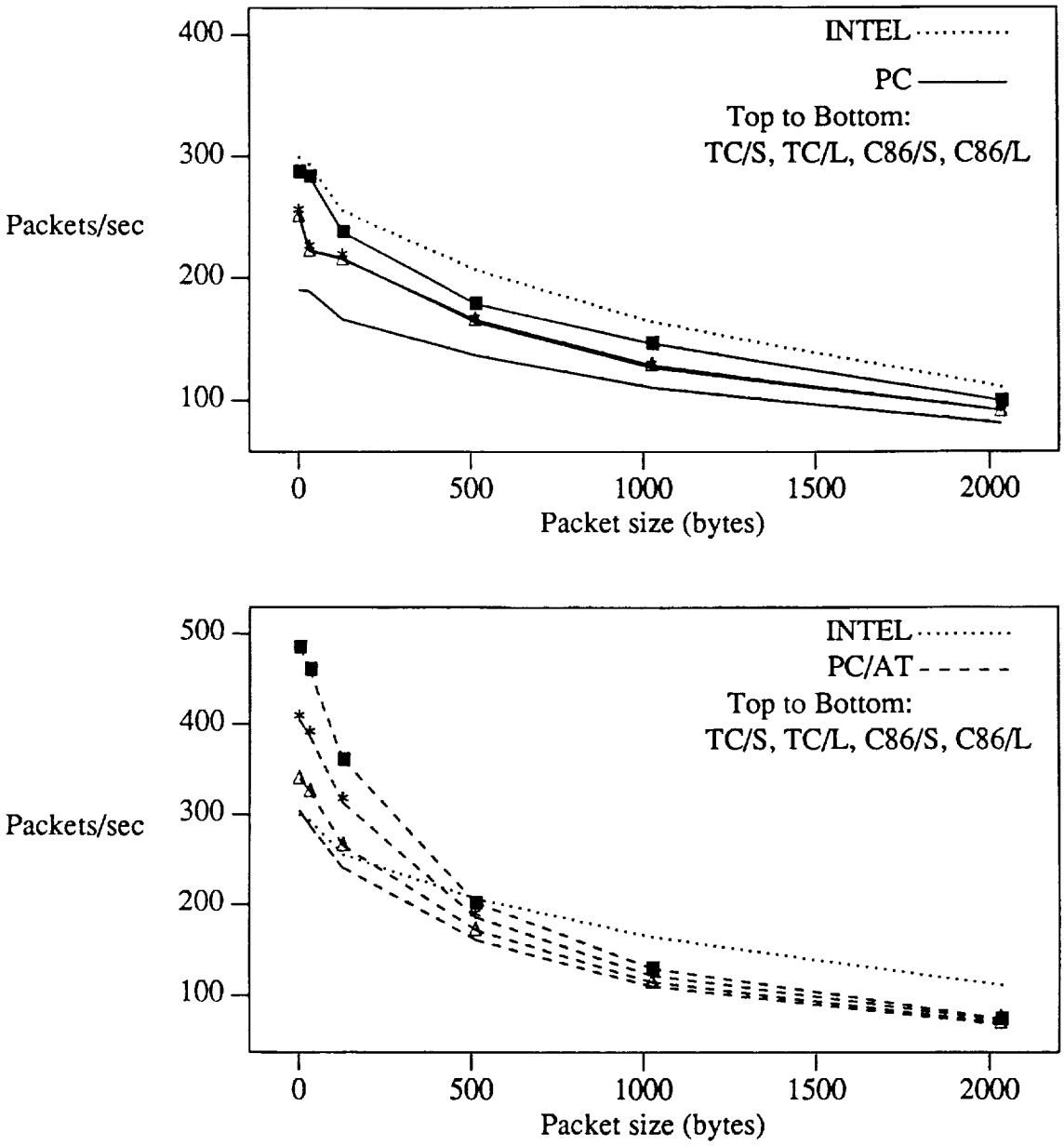


Figure 6
Transmission Rate (packets/sec) vs. Packet Size (bytes)
Showing Effect of Different Compilers

APPENDIX

AIRNET
a realtime communications network for aircraft

Master's Project Presentation

by

Brendan Cain

Background

- Local Area Networks
  - — Resource sharing
  - — File transfer
  - — Distributed real-time control applications
    - Sperry Marine Inc (shipboard control)
    - NASA-Lewis (airframe control)

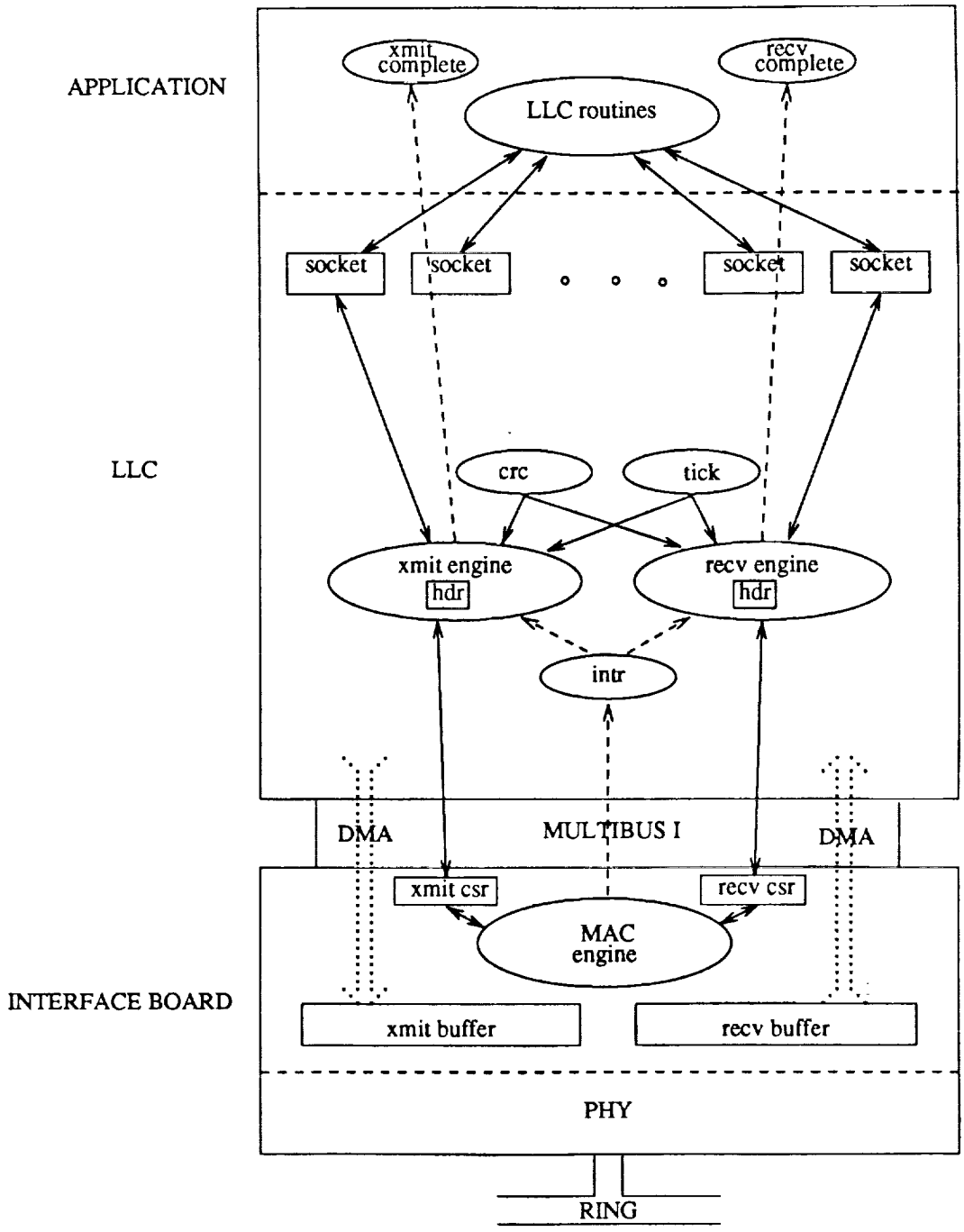
Proteon ProNET-10 Token Ring LAN

- 10 Mbits/sec
- One-byte MAC addresses, and thus 255 nodes (stations)
- Ring or star-ring architecture
- One packet per token
- No priority
- Parity bit error checking
- Bit stuffing to avoid token aliasing

Logical Link Control

- Models IEEE 802.2 Type I and III services
  - — Type I - connectionless, or datagram service
  - — Type III - Type I with immediate packet acknowledgement
- LLC services provided by C library routines
  - — LLCon(sockets)
  - — LLCoff()
  - — LLCopen(socket)
  - — LLCclose(socket)
  - — LLCreset(socket)
  - — LLCstatus(socket)
  - — LLCxmit(socket, station, info, size)
  - — LLCrecv(socket, station, info, size)

Architecture



Project

- Existing system on PCs and PC/ATs
- Produce comparable and compatible system on INTEL 310 (80286)
- Compare LLC performance metrics
  - — Station-generated load (packets/second and bits/second)
  - — End-to-end latency
- Port Test, Measurement, and Application Programs
  - — LLCTEST
  - — LLCWRAP
  - — LLCLOAD
  - — NCP

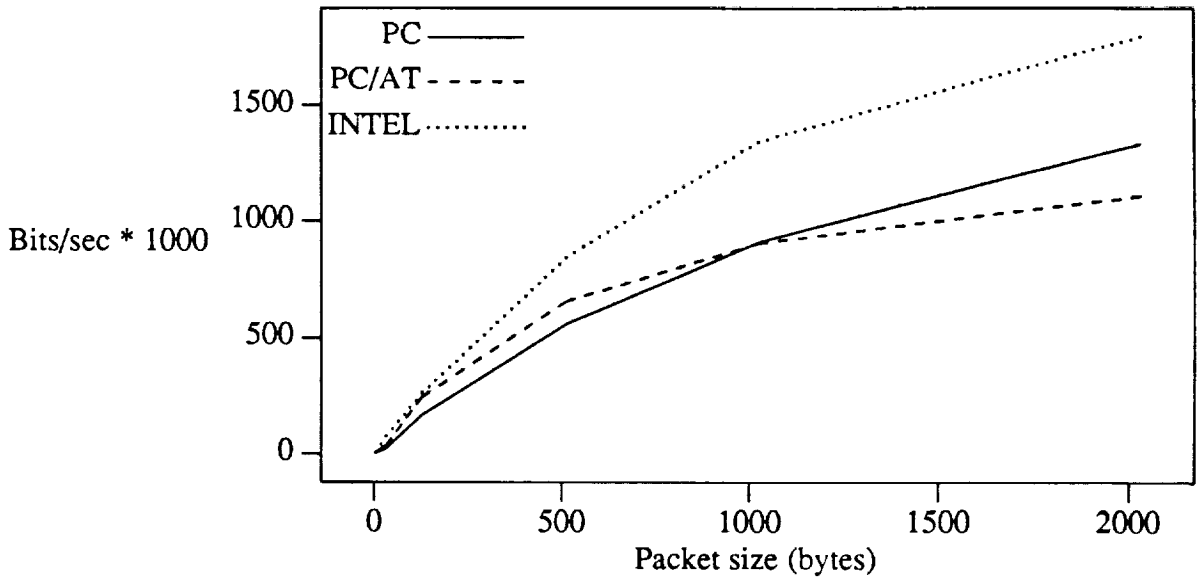
Development Environment

- INTEL 310 (80286) with Multibus I
- RMX86 Operating System
- IC86 Compiler
- Some PL/M Code
- Proteon ProNET-10 p1200
  - — Ring Control Board
  - — Host Specific Interface Board
    - Two 1023 word packet buffers
    - Separate transmit and receive control/status registers
    - Full duplex DMA interface to Multibus

Activities

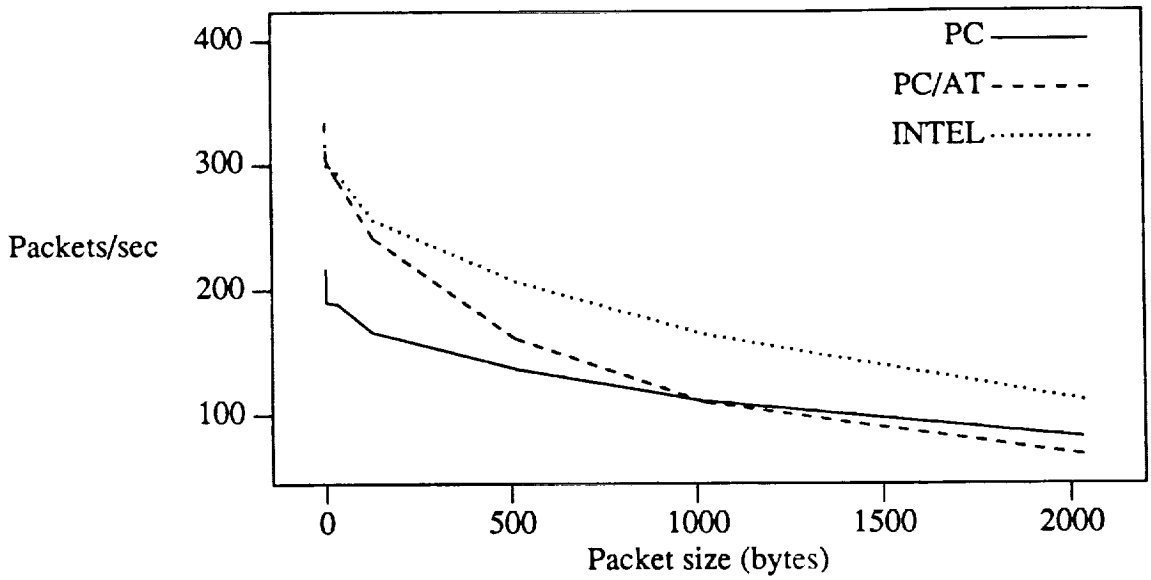
- INTEL machines
- p1200 interface
- Interrupt handling in RMX
- Timing mechanisms
- Packaging

Performance results for PC, PC/AT, and INTEL



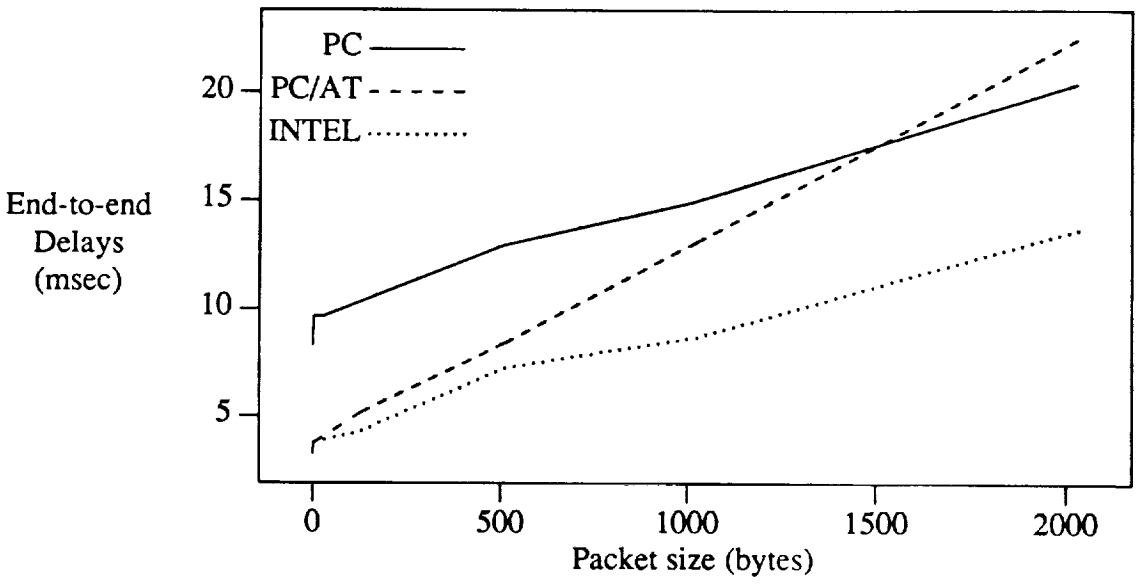
Message length (bytes)	Bits/sec * 1000		
	PC	PC/AT	INTEL
2	3.0	4.9	4.8
32	24.2	37.5	74.9
128	170.2	247.1	260.8
512	560.1	658.4	842.5
1024	909.7	904.9	1341.9
2032	1337.0	1109.9	1801.7

Performance results for PC, PC/AT, and INTEL



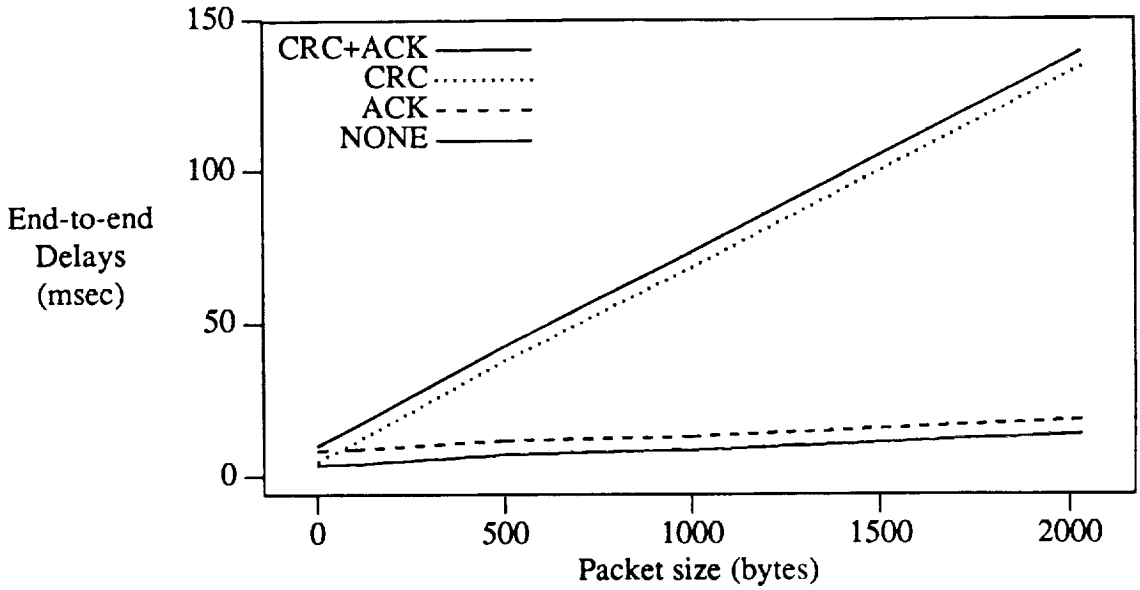
Message length (bytes)	Packets/sec		
	PC	PC/AT	INTEL
0	217	334	315
2	190	304	299
32	189	288	293
128	166	241	255
512	137	161	206
1024	111	110	164
2032	82	68	111

Performance results for PC, PC/AT, and INTEL



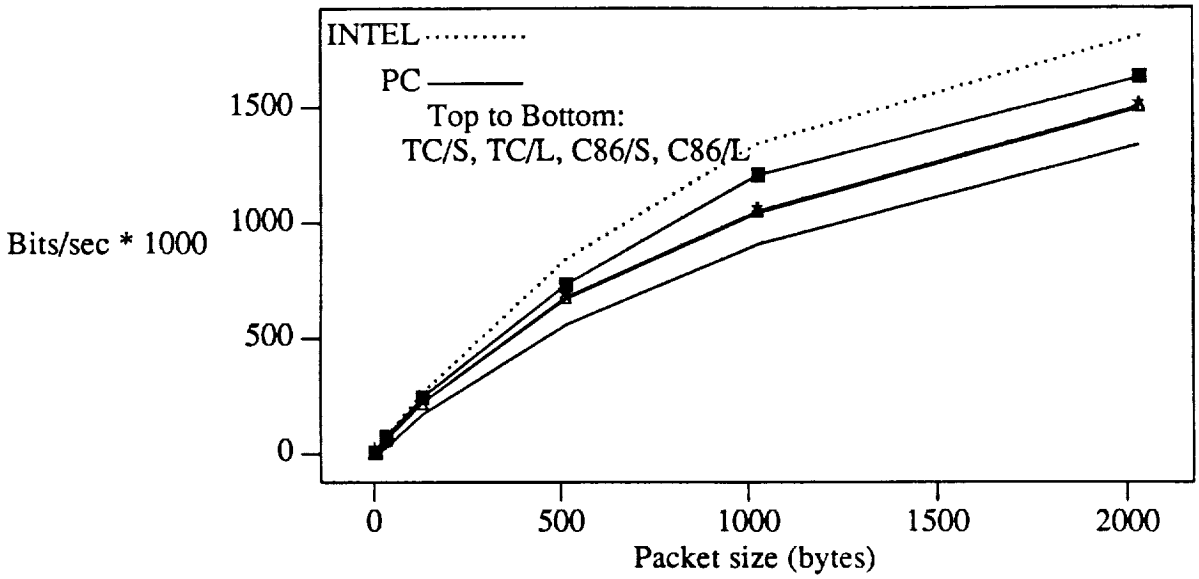
Message length (bytes)	End-to-end delay (milliseconds)		
	PC	PC/AT	INTEL
0	8.283	3.249	3.557
2	9.645	3.752	3.766
32	9.690	-	3.885
128	-	5.191	4.278
512	12.970	8.440	7.240
1024	15.042	13.135	8.707
2032	20.410	22.463	13.669

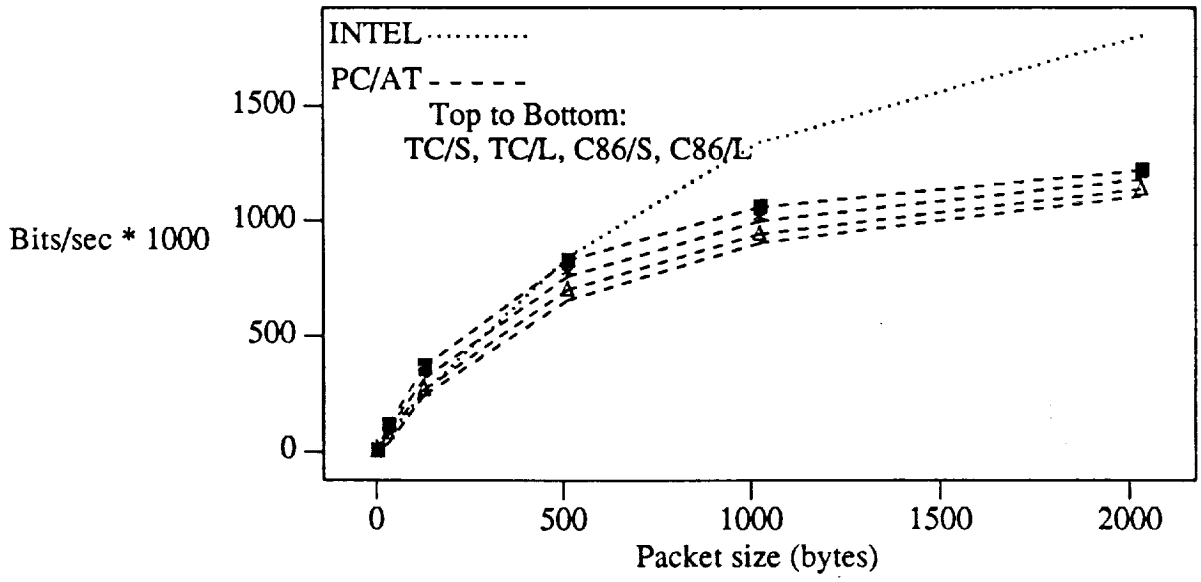
Performance results for INTEL, varying class



Message length (bytes)	End-to-end delay (milliseconds)			
	NONE	ACK	CRC	CRC+ACK
0	3.630	8.303	4.369	9.918
2	3.834	8.513	4.718	10.242
100	4.349	8.961	10.999	16.430
500	7.719	11.864	37.820	42.829
1000	8.583	13.130	67.997	73.405
2032	13.745	18.295	133.673	139.044

Station Load (bits/second * 1000)





Observations Regarding Performance

- Interrupt handling overhead is significant (1 msec)
- DMA channel operation is better (both setup and transfer)
- RMX86 OS and IC86 compiler significant
- Performance in acceptable range?

Acknowledgements

- Alf Weaver
- Alex Colvin
- Tim Strayer
- Robert Simoncic
- John DeLaat

DISTRIBUTION LIST

1 - 3	National Aeronautics and Space Administration Lewis Research Center 21000 Brookpark Road Cleveland, OH 44135 Attention: Dr. J. C. DeLaat, MS 77-1 Advanced Control Technology Branch
4 - 5*	National Aeronautics and Space Administration Scientific and Technical Information Facility P. O. Box 8757 Baltimore/Washington International Airport Baltimore, MD 21240
6	Ms. Lorene Albergottie Grants Officer NASA Lewis Research Center Mail Stop 500 - 315 21000 Brookpark Road Cleveland, OH 44135
7 - 8	A. C. Weaver, CS
9	A. K. Jones, CS
10 - 11	E. H. Pancake, Clark Hall
12	SEAS Preaward Administration Files

*One Reproducible Copy

JO#3271:ph