# Analysis of Transport Measurements Over a Local Area Network

*Sharon Heatley*
*Dan Stokesberry*

THE OPEN SYSTEMS INTERCONNECTION (OSI) protocol standards that have been under development since 1979 promise the capability to interconnect systems from different vendors worldwide without developing special hardware or software; however, there is concern whether implementations of these communications protocols can perform well enough to meet the end-to-end delay and throughput requirements of the users of the communications service. In the past, when data communications took place at speeds of between 110 and 9,600 b/s, the communications channel itself was often the bottleneck. Today, the advent of high-speed local area networks and optical fiber networks has moved the bottleneck to the communications processing part of the system. So, instead of being able to achieve 10 Mb/s or 100 Mb/s, the user can at best achieve the maximum throughput and end-to-end delay available from his communications protocol processor. These throughput and end-to-end delays are often disappointing. It has become fashionable to blame the protocol standards for this poor performance, but there has been insufficient evidence to date to determine how much blame should be given to the protocol and how much belongs to a particular implementation of the protocol. In this paper, we identify some of the implementation factors that affect the performance of OSI protocols.

*Today, the advent of high-speed local area networks and optical fiber networks has moved the bottleneck to the communications processing part of the system.*

The relationship between OSI protocols is described by the seven-layer reference model [1] shown in Figure 1. Many implementations exhibit a break between the top three layers and the bottom four layers. The top three layers are often implemented as software that runs on the host processor, while the bottom four layers are often implemented on a front-end processor that is plugged into the backplane of the host computer. These bottom four layers, Transport and below, provide the service of reliable message transmission between Transport users. The Transport layer is the first layer which provides an end-to-end connection through the network. (In a full seven layer stack, the Transport user would be the Session layer.) The performance of implementations of this transmission service is a particularly interesting subject.

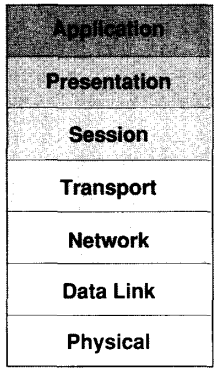


*Fig. 1. OSI protocol stack.*

This paper has two purposes:

- To report the end-to-end delay and maximum throughput for a typical implementation of the bottom four layers so that readers have an idea of the performance currently available
- To identify the factors that affect performance in implementations of these layers of the OSI protocols.

The measurements reported were obtained on Intel 310[1] microcomputer systems. We measured iNA960 release 2 [2], the Intel implementation of the lower four OSI layers (Trans-

[1]Certain commercial equipment is identified in this paper in order to adequately specify the experimental procedure. Such identification does not imply recommendation or endorsement by the National Institute of Standards and Technology, nor does it imply that the equipment identified is necessarily the best available for the purpose.

port, Network, Data Link, and Physical) residing on a 552a front-end communications processor. The user of the Transport services runs on the 286/10 host processor. The 310 nodes are connected via an IEEE 802.3 Carrier Sense Multiple Access with Collision Detection (CSMA/CD) local area network [3]. The end-to-end delay (user-to-user) and maximum throughput for this commercial implementation are typical of the performance currently available. The factors that affect performance are identified, and the relationship between factors are expressed in equations. For throughput, we develop an equation that relates processing time, user message size, and network packet size to throughput. This equation gives the protocol implementor a tool to investigate how a particular change in design (i.e., faster processor, fewer copies of data, larger network packets) will quantitatively change the throughput.

In this paper, we measure the performance of the Intel implementation of Transport Class 4 over a local area network. The performance of the Motorola implementation of Transport Class 4 over a local area network has also been measured [4]. The performance of implementations of Transport Class 4 over a satellite link [5] and over concatenated X.25 networks [6] has also been reported. In addition, there is a study comparing TCP and Transport Class 4 performance over both a local area network and on an X.25 network [7]. This is not intended to be an exhaustive list but simply a few related papers which the reader may find interesting.

## Protocol Overview

The Transport, Network, Data Link, and Physical protocols are described below.

### *Transport*

There are five classes of connection-oriented Transport [8]: Classes 0–4, and there is also a Connectionless Transport [9] protocol. Transport Class 4 offers the most error detection and correction. Connectionless Transport is a "best effort" datagram service, i.e., there is no guarantee of delivery of messages. The implementation we measured, iNA960, contained two Transport protocols: Transport Class 4 and Transport Datagram. Transport Datagram is an Intel proprietary protocol, which offers the same services as Connectionless Transport. Figure 2 shows an overview of the services provided by iNA960. We made measurements of Transport Class 4 and Transport Datagram services and so we will describe both in more detail.

Transport Class 4 offers virtual circuits, error detection and recovery, flow control, segmentation of messages, and in-order delivery of messages. One Transport Class 4 entity communicates with another via Transport Protocol Data Units (TPDUs). The error detection and recovery and flow control functions are accomplished using Acknowledge (AK) TPDUs transmitted by the receiving station. Each Data (DT) TPDU has a field containing a sequence number. Every AK TPDU contains the value of the next sequence number expected by the receiver, thereby acknowledging the receipt of all DT TPDUs with lower sequence numbers. The AK also contains a credit field. The transmitting station is allowed to send DT TPDUs with sequence number $K$ where:

$$AK\ seq.\ no. \leq K < (AK\ seq.\ no. + AK\ credit)$$

This range of permitted sequence numbers is known as the window. There are retransmission timers on the transmitting station, which are restarted when AKs are received. If an AK is not received before the retransmission timer expires, then unacknowledged DT TPDUs are retransmitted. The default retransmission timer value in iNA960 is 500 ms.

The credit returned in AKs in the iNA960 implementation is based on the amount of receive buffer space available at the Transport level. In addition, in the iNA960 implementation, the credit may not be larger than the maximum Transport window parameter (default 15) or less than the minimum Transport credit parameter (default 1). Since the minimum credit is one, each AK always contains permission to send at least one DT so that the window never closes. The Transport Class 4 standard, however, does not specify that an AK must be sent for each DT received. In our iNA960 experiments we observed DT:AK ratios of 1:1 to 5:1.

Transport Class 4 has a maximum TPDU size parameter. The iNA960 default for this parameter is 2,048 octets. The actual TPDU size used is the minimum of the maximum TPDU size Transport parameter and the packet size of the underlying network minus lower layer headers. Transport Class 4 has an optional checksum to guarantee end-to-end data integrity. In all of our experiments, the checksum was turned off.

*In order to understand the results of the experiments, it is important to understand segmentation.*

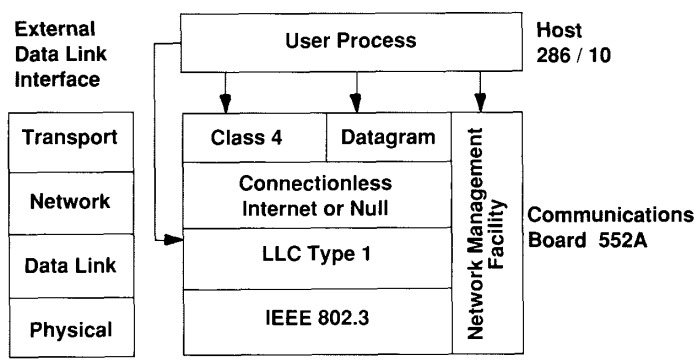


*Fig. 2. Overview of iNA960 services.*

In order to understand the results of the experiments, it is important to understand segmentation. Messages passed by the user to Transport are called Transport Service Data Units (TSDUs). The user may wish to send TSDUs that are too large to send over the communications network in a single packet. An IEEE 802.3 LAN, for example, limits the maximum Data Link frame size to 1,518 octets. One of the services offered by Transport Class 4 is to segment into multiple TPDUs those TSDUs which are too large to fit into network packets, where each TPDU fits into a single network packet. Thus, the user of Transport Class 4 can pass arbitrarily large TSDUs to Transport without knowledge of the network packet size.

Transport Datagram and Connectionless Transport do not segment messages, so that users of these services must ensure

that the size of the message is smaller than the maximum which can be sent on the network. These protocols do not guarantee delivery and also do not guarantee that messages are delivered in order. The protocols therefore do not use acknowledgments or timers.

### *Network*

A Connectionless Internet Protocol (IP) [10] standard has been developed for the Network layer. IP has the responsibility for routing and relaying packets in an Internet environment, i.e., different networks concatenated together. When the iNA960 software is configured, the Network protocol can be chosen to be either Connectionless IP or the null IP header. If the null IP header is chosen, then no IP processing is done. Since all experiments were done on a single LAN, the IP functions are not needed; but we did experiments with both options in order to determine the additional overhead added by this implementation of the IP layer.

### *Data Link*

For local area networks, the Data Link layer is divided into two sublayers. The top sublayer is the Logical Link Control (LLC) [11]. LLC type 1 provides datagram service, while LLC type 2 provides a connection-oriented service. The bottom sublayer is the Medium Access Control (MAC) layer. There is a family of MAC protocols—IEEE 802.3 (CSMA/CD) and IEEE 802.4 [12] (token bus) are members. FDDI [13] is another example of a MAC protocol.

For these experiments, connectionless LLC type 1 and the IEEE 802.3 CSMA/CD MAC protocol are used. In the iNA960 implementation, the LLC protocol has an external interface so that an application running on the host board may directly request LLC datagram service without using the services of either Transport or Network. This service, which we will call External Data Link service, is like the Transport Datagram service in two respects: it does not guarantee delivery and it does not segment messages. The messages passed by the user to the LLC layer are called Link Service Data Units (LSDUs).

### *Physical*

The Physical layer protocols specify how the individual bits are to be sent on the physical channel. The IEEE 802.3, 802.4, and the FDDI [14] [15] standards specify the Physical layer protocols.

## Experimental Environment

### *Hardware/Software Overview*

The Intel 310 microcomputer system, as shown in Figure 3, includes an Intel 286/10 host board, 1 Mbyte of memory, and an Intel 552a communications front-end board connected together by the Intel Multibus. The host board contains an 80286 processor running at a clock rate of 6 MHz. In our experiments, the 286/10 host runs the traffic generation software, which uses the communication services of the front-end board and also gathers statistics on performance. The 286/10 hosts in all systems are connected to a central clock developed at NIST. This clock makes it possible to measure end-to-end delay to within 0.1 ms.

The 552a board runs the iNA960 communications software, which is an implementation of the lower four OSI layers. This board contains an 80186 processor running at a clock rate of 8 MHz, an 82586 LAN coprocessor, an 82501 serial interface, and 256 kbytes of local memory. The 80186 executes code that implements the Transport, Network, and Data Link (LLC Type 1) protocols. The 82586 LAN coprocessor executes the IEEE 802.3 CSMA/CD MAC protocol. It transmits and receives data over the IEEE 802.3 medium through the 82501 serial interface. The 286/10 host and the 552a front-end communicate, using the Multibus Interprocessor Protocol (MIP), by writing messages into the Multibus memory and then interrupting the processor on the other board to indicate that a message has been written.

The host and front-end can operate independently. The front-end board works on the host's communication service requests while the host does other processing. On the front-end board, the 80186 executing the Transport, Network and Data Link protocols and the 82586 LAN coprocessor can also operate independently. Each 310 system contains three independently operating processors. When there is a transmission between two systems, six independently operating processors are involved.

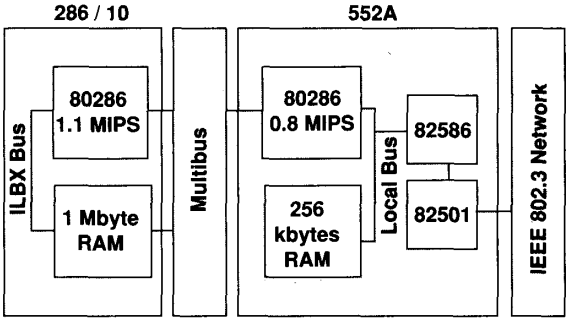


*Fig. 3. Intel 310 system overview.*

### *Buffer Management*

The iNA960 implementation reserves a number of Data Link buffers in the 256-kbyte local memory on the 552a board. The size of each transmit buffer is 1,500 octets. This is large enough to store the memory image of a full IEEE 802.3 frame. Receive buffers are 256 octets each; these buffers are chained together to store the memory image of a received Data Link frame. The number of transmit and receive buffers reserved are configuration parameters. The default number of transmit buffers is 5 and the default number of receive buffers is 255.

User processes on the host requesting Transport services or Data Link services from iNA960 allocate data buffers in Multibus memory. When the user sends iNA960 a request to transmit data, the request contains a pointer to an associated Multibus data buffer that contains the user message (TSDU or LSDU). The data in the user data buffer is copied by the iNA960 software into one or more Data Link transmit buffers on the 552a board. More than one Data Link transmit buffer is used if Transport Class 4 segments the TSDU into more than one TPDU. The data is then sent out on the LAN by the 82586 LAN Coprocessor. On the receiving side, the same set of events happens in reverse. The 82586 LAN coprocessor receives the data and stores it in Data Link receive buffers. The data in the Data Link receive buffers is copied into user receive buffers previously posted by the host user process. When the buffers are full or when they contain a complete user message, they are returned to the user on the host.

## Experimental Results

Unless otherwise noted, iNA960 default values are used for all parameters in these experiments.

### *Delay Measurements*

For the first set of experiments, we report the end-to-end delay for Transport Class 4, Transport Datagram, and External Data Link services. This delay is measured from transmitting host user process to receiving host user process and is the minimum delay that can be achieved when one message is transmitted in an idle system (i.e., there are no queuing delays anywhere in the system). We also compare Transport delays measured when using IP with those obtained when using the null IP header in order to determine the additional delay associated with this implementation of the IP protocol.

*We also compare Transport delays measured when using IP with those obtained when using the null IP header in order to determine the additional delay associated with this implementation of the IP protocol.*

Figure 4 shows one-way delay versus message size for Transport Class 4 with a Null IP header (TP4/Null), Transport Datagram with a Null IP header (TPD/Null), and External Data Link (EDL) services. The range of message sizes is 16 to 1,500 octets. For these message sizes, all user messages fit into a single network packet. The curves in Figure 4 can be described by straight lines of the form $T = m * x + b$, where $T$ is the delay in milliseconds and $x$ is the message size in octets. The best estimates for $m$ and $b$ are:

**TABLE I.**

| | m (ms/octet) | b (ms) |
|---|---|---|
| **TP4/Null** | **0.0028** | **10.7** |
| **TPD/Null** | **0.0028** | **9.3** |
| **EDL** | **0.0045** | **6.6** |

The $y$-intercept (i.e., $b$) for each line is the amount of delay that is invariant to the size of the message. This includes:

- Communications between the host board and its communications board at the sending side
- The processing on the communications board needed to send a message
- The processing on the receiving communications board needed to receive a message
- Communication between the communications board and the host board at the receiving side

The slope of each line (i.e., $m$) is the amount of processing that depends on the message size. This processing includes the copy from Multibus memory to local memory on the communications board (which is done under the control of the communications CPU), transmission of the message, and the copy from local memory to Multibus memory on the receive side. For Transport Class 4 with IP (TP4/IP) and Transport Datagram with IP (TPD/IP), the slope of the lines are the same

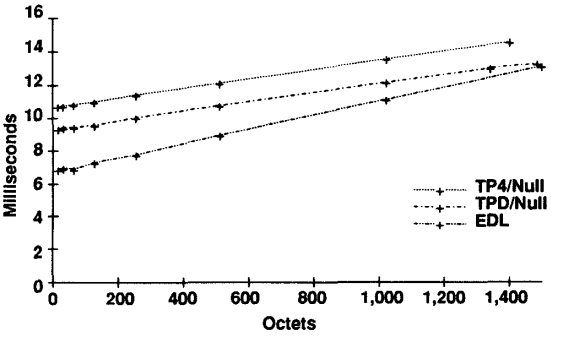


*Fig. 4. One-way delay vs. message size.*

as for TP4/Null and TPD/Null, but the $y$-intercept is one millisecond more. The IP layer adds about an additional millisecond of delay.

Figure 5 shows one-way delay versus message size for TP4/IP and TP4/Null. The range of message sizes is from 16 to about 11,000 octets. As noted above, Transport Class 4 can receive arbitrarily large messages from the user and segment them into TPDUs, each fitting into one network packet. The front-end does a certain amount of processing for each TPDU sent or received. This processing causes the jumps in the delay curves that occur at the points where the TSDU or user message is segmented into an additional TPDU. Since the data link level buffers are 1,500 octets long, each TPDU must fit into a 1,500 octet network packet. When there is a null IP header, 1,488 octets of Transport user data (and 12 octets of Transport header) fit into one TPDU. When there is an active IP layer, 1,401 octets of Transport user data fit into one TPDU. The number is smaller for the IP case because of the space required for the IP header. Thus, for the Transport Class 4 with an IP layer, a jump in one way delay occurs between 1,401 and 1,402-octet messages since a 1,401-octet message fits into one TPDU and is sent in one network packet but a 1,402-octet message must be segmented into two TPDUs and sent in two network packets. A similar jump occurs between $1401 * k$ and $(1401 * k) + 1$ where $k = 2, 3....$ The average jump in delay for processing an additional TPDU for $k = 1...7$ is about 2.5 ms for TP4/IP and 2.0 ms for TP4/Null.

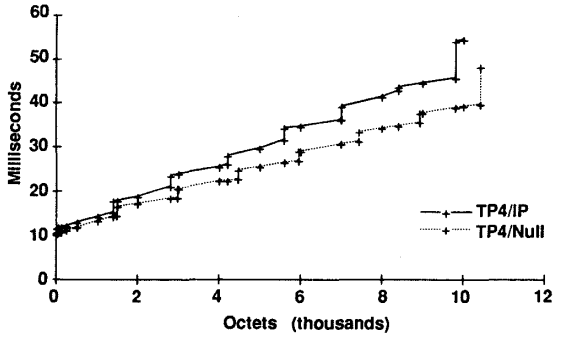


*Fig. 5. One-way delay vs. message size.*

### *Throughput Experiments*

Throughput is reported for TP4/IP and TP4/Null. Throughput is defined as the total number of octets sent divided by the time required to send them. Time is measured from when the

first TSDU is sent by the sending user process until the last TSDU is received by the receiving user process. The throughput measurements in this paper were done for memory-to-memory bulk transfer applications. In these experiments, the traffic generation process on the host boards allocate a pool of one or more user transmit buffers on the transmit side and a pool of one or more user receive buffers on the receive side. Each buffer contains one user message. As soon as the data in a user transmit buffer is acknowledged at the Transport level, the transmit buffer is returned to the traffic process on the host. This process immediately returns the buffer to iNA960 with another transmit request. Thus, the pool of user transmit buffers circulates between iNA960 and the traffic process. In a similar way, the user receive buffers circulate between iNA960 and the receiving traffic process. If there is more than one buffer in the transmit or receive buffer pool then host processing and communications processing can be overlapped.

Experiments were done to determine the values of Transport Class 4 parameters which resulted in the best throughput for a memory-to-memory bulk data transfer. The parameters which were "tuned" for the bulk data transfer situation were number of user transmit buffers, number of user receive buffers, maximum window size, and minimum retransmission timer setting. For the maximum window size and minimum retransmission timer size, the iNA960 default values of 15 and 500 ms are adequate for maximum throughput. At least two transmit buffers and three receive buffers are required for maximum throughput. (No transmissions errors or dropped packets occurred in our experiments. The retransmission timer has no effect on performance unless it is too short.)

Figure 6 shows throughput versus message size for TP4/IP and TP4/Null. Message size varies from 100 octets to 25,000 octets. For all experiments reported in this section, there are 10 user transmit buffers and 10 user receive buffers. For each user message, a certain amount of processing is required to pass the message from the user process to iNA960 at the transmitting side and from iNA960 to the user process at the receive side. With 100 octet messages, this communication has to be done 10,000 times to send one Megabyte of data. With 10,000 octet messages, this communication has to be done 100 times to send one Megabyte of data. Reducing this processing causes throughput to increase and, therefore, as the user message size increases, the throughput generally goes up.

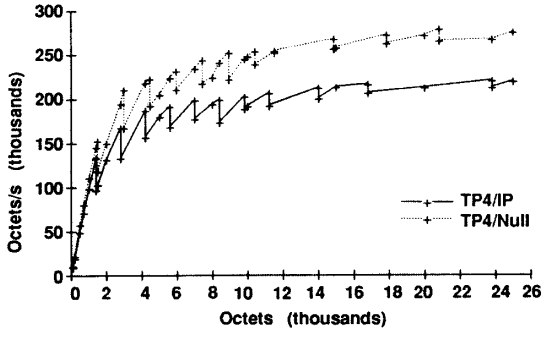


*Fig. 6. Throughput vs. message size.*

There is another factor that increases throughput as the user message size increases from 100 octets to the maximum number of user octets in a TPDU (1,401 octets for TP4/IP, 1,488 for TP4/Null). To send one Megabyte of data, with 100 octet user messages, 10,000 TPDUs are sent since each user message must be sent in a separate TPDU. To send one Megabyte of data with 1,401-octet user messages, only 714 TPDUs are sent. This reduces the amount of TPDU processing and increases throughput.

The throughput drops sharply whenever the user message is segmented into an additional TPDU, resulting in additional processing for each user message. For example, for TP4/IP, when the message size is 1,401, one TPDU is transmitted for each user message. When the message size is 1,402, two TPDUs must be transmitted for each user message although the second TPDU contains only one octet of user data. The drop in throughput shown in Figure 6 is due to the additional time required to process the second TPDU. The maximum throughput measured with TP4/IP and a user message size of 25,000 octets is 220 kbytes/s (1.76 Mb/s). The throughput for TP4/Null and a user message size of 25,000 octets is 276 kbytes/s (2.2 Mb/s). This difference in throughput is due to the extra processing time required to execute the code that implements the IP protocol.

In a previous paper we developed simple models which relate user message size, processing times, and maximum user data contained in a TPDU to throughput and messages per second [16]. These models assume that either the sending or receiving 80186 (the front-end which processes the protocols Transport Class 4, IP, and LLC type) is the bottleneck in the system. The other two assumptions made are that the bottleneck processor is continuously busy and that no errors or retransmissions occur. The models are shown below:

$$THR(x) = x/(A * x + B * ceil(x/M) + C)$$

and

$$MPS(x) = 1/(A * x + B * ceil(x/M) + C)$$

where $x$ is the user message size, $THR(x)$ is throughput in octets per second achieved for user message size $x$, $MPS(x)$ is the number of messages per second achieved for user message size $x$, *and* $M$ is the maximum amount of user data that can be contained in a TPDU.

$Ceil(x/M)$ is the smallest integer greater than $x/M$. Therefore, $ceil(x/M)$ is the number of TPDUs required to transmit a user message of size $x$. $A$, $B$, and $C$ are processing times on the bottleneck processor. $A$ is the processing done per octet. This processing includes the copy between the local memory and the Multibus memory, which is done under the control of the communications processor and may include the overhead associated with the copy between the local memory and the FIFO in the 82586 coprocessor. Transport checksum is OFF for these experiments, so there is no checksum processing per octet. $B$ is the processing done for each TPDU in the Transport, IP, and LLC layers (including AK processing). $C$ is the processing that occurs on a per user message basis, such as the communication between the host and the front-end.

By using curve fitting techniques, we can fit the throughput model to the measured data shown in Figure 6 and estimate the parameters $A$, $B$, and $C$ [16].

**TABLE II.**

| | A (ms/octet) | B (ms/TPDU) | C (ms/message) |
|---|---|---|---|
| **TP4/IP** | **0.0014** | **4.11** | **4.75** |
| **TP4/Null** | **0.0013** | **3.14** | **4.86** |

Figure 7 shows the live throughput data and the estimated data using the above equation for TP4/IP and TP4/Null. The difference between the live and estimated data is less than 7%.

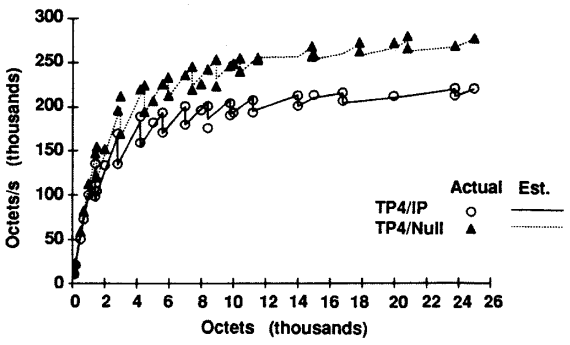


*Fig. 7. Throughput vs. message size.*

Now that we have this throughput equation, we can see how sensitive throughput is to changes in the parameters *A, B, C,* and *M*. Figure 8 shows the effects on throughput of keeping the processing per TPDU and the processing per message constant and varying the amount of processing per octet. This illustrates the importance of minimizing the number of copies required. Figure 9 shows the effects on throughput of keeping the processing per message and the processing per octet constant and varying the processing per TPDU. In our protocol overview section, we noted that in iNA960, the DT:AK ratio varied from 1:1 to 5:1. One way to decrease the processing per TPDU is to increase the DT:AK ratio since AK processing is included in B. Figure 10 shows the effects on throughput of keeping the other processing the same and varying the amount of processing per user message. When user message size is small in Figure 10, the effect on throughput of varying the user message processing can be substantial. As the user message size increases, more TPDUs and more octets are transmitted for each user message. The processing time per TPDU and per octet then dominate the throughput. The effect of varying the user message processing time becomes negligible for large user message sizes.

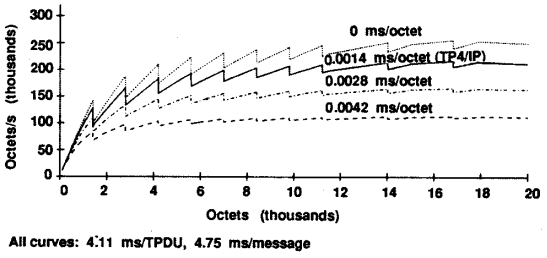


*Fig. 8. Throughput vs. processing per octet.*

Experiments were done in which the maximum TPDU size (*M*) was varied and throughput was measured. In these experiments, the user message size is 10,000 bytes. In Figure 11, measurements are compared with the data predicted by the throughput equation for throughput versus maximum user data per TPDU. Since the IEEE 802.4 token bus standard has a maximum packet size of 8,018 octets, the graph has been extended out to 8,000 octets of user data per TPDU. If we can make the same assumptions in a token bus system as for an IEEE 802.3 system, then extending the graph in this way seems

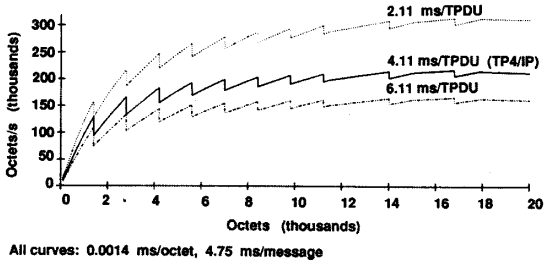


*Fig. 9. Throughput vs. processing per TPDU.*

reasonable. These assumptions are that there is a token bus IEEE 802.4 MAC processor which operates independently from the 80186 processor and that the Transport, IP, and LLC processing on the 80186 is the bottleneck in the system. We also assume there are no retransmissions (and, in this case, no lost tokens) and that the bottleneck processor is continuously busy. If these assumptions are met, then a maximum user data per TPDU of 8,000 octets on a token bus network would allow a throughput of about 350,000 octets/s for user message sizes of 10,000.

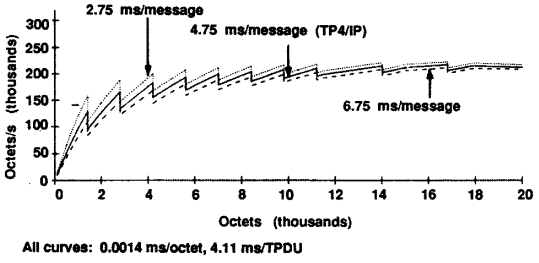


*Fig. 10. Throughput vs. processing per user message.*

As $x$ (the user message size) becomes larger, the throughput approaches $1/(A + B/M)$. For the case of TP4/IP, this maximum throughput for very large message sizes is calculated to be 230,947 octets/s. As shown in Figure 6, the measured throughput approaches this line. If we substitute 8,000 for $M$ in the equation above, we get a maximum throughput of about 500,000 octets/s or 4 Mb/s for a token bus network.

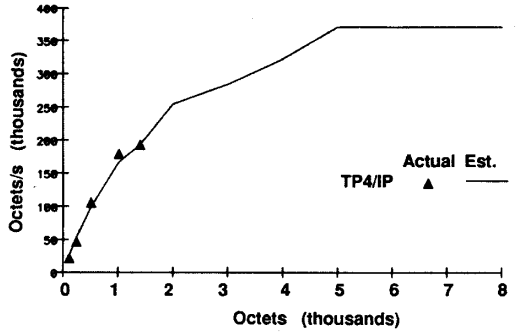


*Fig. 11. Throughput vs. maximum user data per TPDU.*

Figure 12 shows the excellent correlation between the equation for messages per second and the measurements for the case of TP4/IP and TP4/Null. The maximum number of messages per second occurs as the message size approaches zero. The number of messages per second then approaches $1/(B + C)$. For TP4/IP, this gives a maximum message rate of 113 messages/s and, for TP4/Null, 125 messages/s. The measured message rate for 100 byte messages is 100 messages/s for TP4/IP and 122 messages/s for TP4/Null.

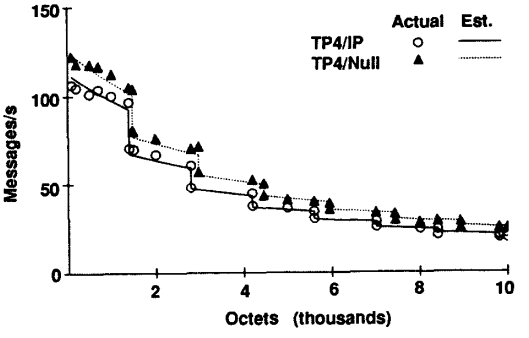


*Fig. 12. Message rate vs. message size.*

# Conclusions

As noted in the introduction, there is concern whether implementations of OSI communication protocols can meet the low end-to-end delay and high throughput requirements of certain military and industrial users. It is important to note that all transport level protocols offering the same services as Transport Class 4 have certain implementation problems in common: timer management, buffer management, connection state management, transfer of data from the user, division of the protocol processing into processes, interprocess communication, and scheduling. It can be seen that the choices that are made in solving these problems affect protocol processing times and thus affect throughput and end-to-end delay. In addition, the speed of the processor, memory and bus on which the protocol is implemented, whether it is a host or front-end implementation and the cost of the host/front-end communication also affect protocol processing times. We also saw in our experiments that the user message size and network packet size can dramatically affect throughput.

In general, there is a need for better performance than that available from current implementations of communications protocols. This improvement can be obtained by either inventing better protocols or by improving implementations of the present standard OSI protocols. It is clear that inventing new protocols by itself is not sufficient because the choices made in implementing the protocol have a dramatic effect on performance, whether the protocol is an OSI protocol or some other protocol. When deciding whether OSI protocols will meet a set of performance requirements, it is important to examine available performance data and to also carefully examine the implementations that produced the data; otherwise, there will be a tendency to eliminate the use of standard protocols and lose all their benefits just because the measurements were made on a particular implementation which does not meet the user's requirements.

# References


[1] International Organization for Standardization, *Open Systems Interconnection—Basic Reference Model*, ISO/TC 97/SC21, International Standard ISO/IS 7498 rev., 1984.

[2] Intel, *iNA960 Release 2 Programmer's Reference Manual*, Jan. 1987.

[3] *Carrier Sense Multiple Access with Collision Detection*, IEEE Standard 802.3, 1984.

[4] W. T. Strayer and A. C. Weaver, "Performance Measurement of Motorola's Implementation of MAP," *13th Local Computer Networks Conference*, Minneapolis, MN, Oct. 1988.

[5] R. Colella, R. Aronoff, and K. Mills, "Performance Improvements for ISO Transport," *Proceedings: Ninth Data Communications Symposium*, Sept. 1985.

[6] R. Colella, J. Fox, P. Markovitz, and L. Gebase, "OSI TP/IP Over X.25: A Performance Study," *USENIX*, June 1987.

[7] Cole, R. and Lloyd, P., "OSI Transport Protocol—User Experience," *Open Systems '86: Online Publications*, Pinner, Middx, UK, 1986.

[8] International Organization for Standardization, *Transport Protocol Specification*, ISO/TC 97/SC 16. Draft International Standard ISO/DIS 8073 rev., June 1984.

[9] International Organization for Standardization, *Protocol for Providing the Connectionless-mode Transport Service*, ISO/TC 97/SC 6. Draft International Standard ISO/DIS 8602 rev., June 1987.

[10] International Organization for Standardization, *Protocol for Providing the Connectionless-mode Network Service*, ISO TC 97/SC 6. Draft International Standard ISO/DIS 8473 rev., Mar. 1986.

[11] *Logical Link Control*, IEEE Standard 802.2, 1984.

[12] *Token-Passing Bus Access Method*, IEEE Standard 802.3, 1985.

[13] American National Standard, *Fiber Distributed Data Interface (FDDI) Token Ring Media Access Control (MAC)*, X3.139-1987.

[14] Draft Proposed American National Standards, *Fiber Distributed Data Interface (FDDI), Token Ring Physical Layer Medium Dependent (PMD)*, X3.166.198X.

[15] American National Standard, *Fiber Distributed Data Interface (FDDI), Token Ring Physical Layer Protocol (PHY)*, X3.148-1988.

[16] S. Heatley and D. Stokesberry, "Measurements of a Transport Implementation Running Over an IEEE Local Area Network," *Proceedings: Computer Networking Symposium*, Washington, D.C., Apr. 1988.


# Biography


**Sharon Heatley** earned an M.A. in mathematics and an M.S. in computer science from the University of Wisconsin in Madison. She previously worked on switching systems at Bell Laboratories and CAT scanners at General Electric Medical Systems. For the past four years, she has done research on the performance of OSI protocols at the National Institute of Standards and Technology.

**Dan Stokesberry** obtained a B.S. in physics in 1962 and an M.S.E.E. from the University of Maryland in 1970. He has worked at the National Institute of Standards and Technology since 1962. He is currently Manager of the Network Management Group in the Systems and Network Architecture Division.