Performance Evaluation of the ARPANET Transmission Control Protocol in a Local Area Network Environment

Samuel T. Chanson, K. Ravindran & Stella Atkins

July 1984
Revised December 1984
Technical Report 85-6*

ABSTRACT

The Transmission Control Protocol (TCP) of ARPANET is one of the most popular transport level communication protocols in use today. Originally designed to handle unreliable and hostile subnets in a long-haul network TCP has been adopted by many local area networks (LAN) as well. It is, for example, available in 4.2 BSD UNIX for interface to Ethernet and several other LAN technologies. This is convenient but not desirable from a performance standpoint since the control structure is far more complex than is necessary for LANs.

This paper describes what we learned in measuring and tuning the performance of TCP in transferring large files between two hosts of different speeds over the Ethernet. Models are presented which allow the optimal buffer size and the flow control parameter to be determined. Based on observed traffic patterns and those reported elsewhere, we formulated guidelines for the design of transport protocols for a single LAN environment. We then present a new, much simpler LAN transport level protocol which replaces TCP with significant improvement in network throughput. Internet packets will use the full TCP. This is done at the gateway. Since the majority of the packets in a LAN are for local usage, this scheme improves the overall network throughput rate as well as the mean packet delay time.

* Also appears in INFOR vol.23, no.3, August 1985. This work was supported in part by the Natural Sciences and Engineering Research Council of Canada.

PERFORMANCE EVALUATION OF THE ARPANET TRANSMISSION CONTROL PROTOCOL IN A LOCAL AREA NETWORK ENVIRONMENT*

SAMUEL T. CHANSON, K. RAVINDRAN, AND STELLA ATKINS

Department of Computer Science, University of British Columbia, Vancouver, B.C., Canada

ABSTRACT

The Transmission Control Protocol (TCP) of ARPANET is one of the most popular transport level communication protocols in use today. Originally designed to handle unreliable and hostile subnets in a long-haul network, TCP has been adopted by many local area networks (LAN) as well. It is, for example, available in 4.2 BSD UNIX for interface to Ethernet and several other LAN technologies. This is convenient but not desirable from a performance standpoint, since the control structure is far more complex than is necessary for LANS. This paper describes what we learned in measuring and tuning the performance of TCP in transferring large files between two hosts of different speeds over the Ethernet. Models are presented that allow the optimal buffer size and the flow control parameter to be determined. Based on observed traffic patterns and those reported elsewhere, we formulated guidelines for the design of transport protocols for a single LAN environment. We then present a new, much simpler LAN transport level protocol which replaces TCP with significant improvement in network throughput. Internet packets will use the full TCP. This is done at the gateway. Since the majority of the packets in a LAN are for local usage, this scheme improves the overall network throughput rate as well as the mean packet delay time.

RÉSUMÉ

La procédure de transmission TCP de ARPANET est aujourd'hui l'unes des procédures les plus populaires pour le niveau de transport. A l'originée destinée à des sous-réseaux d'un fonctionnement incertain de réseaux à longue distance, elle a été depuis adoptée par de nombreux réseaux locaux (LAN). Par exemple, UNIX 4.2 BSD se sert de TCP pour l'interface avec l'Ethernet. Cela est commode, mais la performance est médiocre, parce que la structure de contrôle est beaucoup plus complexe qu'il n'est nécessaire pour un réseau local. Nous décrivons ce que nous avons appris en mesurant et ajustant la performance de TCP pour le transfert de large fichiers entre deux processeurs centraux de vitesses différentes avec un Ethernet. Nous présentons une nouvelle procédure de transmission, beaucoup plus simple que TCP, pour le niveau de transport local. Les paquets pour longues distances utilisent TCP, à partir de la porte du réseau. Comme la majorité des paquets dans un LAN est pour usage local, ceci améliore le volume du réseau, ainsi que le délai moyen par paquet.

1. INTRODUCTION

The Transmission Control Protocol (TCP) of ARPANET is one of the most popular transport level communication protocols in use today. Originally designed to handle unreliable and hostile subnets in a long-haul network, TCP has been adopted by many local area networks (LAN) as well. It is, for example, available in 4.2 BSD UNIX for interface to Ethernet[24] and several other LAN technologies. This is convenient but not desirable from a performance standpoint, since the control structure is usually much more complex than is necessary for LANS.

*Received July 1984; revised December 1984

Unlike long-haul networks (LHN), LANS are characterized by high channel speed (typically 10Mb/sec as opposed to 9.6Mb/sec for LHNS), low transmission error rates, and a controllable environment. The major portion of the packet delay time (time to process and successfully deliver a packet to its destination) shifts from the propagation delay time for LHNS to protocol processing time for LANS. Thus the efficiency of high-level protocols is an important design issue for local area networks.

The Department of Computer Science at the University of British Columbia runs 4.2 BSD UNIX on several VAXES and SUN (MC68010-based) workstations connected by a 10Mb/sec Ethernet. This paper describes what we learned in measuring and tuning the performance of TCP in transferring large files between a fast host (VAX11/750) and a slower host (SUN workstation) over the Ethernet. We then present a new, much simpler transport-level protocol which replaces TCP with significant improvement in network throughput. When packets are intended to be sent to other networks which may be unreliable and which support only TCP, the full TCP can be used. This is done at the gateway. Since the majority of the packets in a LAN are for local usage, this scheme greatly improves the network throughput rate as well as the mean packet delay time.

2. TYPICAL LAN TRAFFIC

Messages generated in a local area network environment fall into three major categories.[14]

2.1 Remote service request – reply type i/o

A client level process initiates a service request for a remote server and waits for a reply. The server processes the request and sends a reply messge. This sequence of activities is characterized by the exchange of short bursts of variable length messges.[15] This type of exchanges are best supported by a datagram protocol like the UDP (User Datagram Protocol) or one that is custom-designed, such as the i/o protocol used in the V-System[8] and the remote IPC protocol used in SHOSHIN.[9] Common applications that generate such exchanges include remote logins, remote executions, remote procedure calls, and remote file accesses.

2.2 System-generated messages

Some messages are not explicitly initiated by a client level transaction but are generated by the system as a consequence of such a transaction. An example is a file creation/deletion request. This process necessitates propagation of directory updates to the various workstations maintaining in-core directory information. Other messages are unrelated to client activities such as event notifications, acknowledgements and status exchanges.[16,18] These messages are short (mostly one or two packets) and infrequent. This type of messages also calls for a connectionless protocol as the packet transport mechanism.

2.3 Stream type messages

These require the support of a connection oriented protocol, and are characterized by a unidirectional transfer of much bigger packets and connections of longer life time than the other types of traffic.[14] The overhead due to initial connection set-up and run-time connection management common in virtual circuit protocols is justified by the efficiency provided. Typical applications generating this type of traffic are large file transfers, process control applications, image, and voice traffic. Throughput rate is the most important performance consideration for file transfer type applications while delay times are more crucial in real-time closed loop type activities and image/voice data traffic. TCP/IP is a typical protocol used to support this type of applications which create much more demand on network bandwidth than the other applications (IP stands for Internet protocol and is a network level protocol). Later sections describe the performance evaluation of TCP/IP in supporting such stream type traffic, and the viability of a custom-designed protocol to replace TCP/IP to handle primarily this but also the other types of traffic.

2.4 Some empirical data on LAN load characteristics

Knowledge of the characteristics of the network traffic as well as the underlying medium under operational loads is valuable in protocol design. Some measurement data are available,[16,17,18,19,23] and are summarized below. Although all the data had been collected from Ethernet installations, with the exception of the Ethernet specific statistics such as packet collision rates, most apply to other types of LANs as well.

1. The peak load even for a large LAN typically constitutes only a small fraction of the network capacity.[16,17,18,23] Measured figures on Ethernet[16] (presumably using PUP internet protocols[25]) in a single LAN consisting of 120 Alto machines, two time-sharing systems (TENEX), gateways, file and print servers, etc. showed that the maximum load as observed over a six-minute interval was 7.9% and the average load

0.8% of the Ethernet capacity of 2.94 Mb/sec. The corresponding figures over an observation interval of one second were 32.4% and 2.7% respectively. Reference [17] reported measurement data on an existing university time-sharing environment extrapolated to that for a large scale Ethernet-based LAN supporting the same environment. The network utilization for 1,000 users with heavy network disk usage was no more than 30%.

2. For a given total load level, network performance is largely insensitive to the configuration.[16] For example, five stations, each presenting 20% of the given load onto the network, have largely the same effect as fifty stations each inputing 2% of the load.
3. Packet arrivals tend to be very bursty.[16,18,19]
4. Packet lengths exhibit a bimodal distribution[16,18] characterizing the message types discussed in sections 2.1 to 2.3.
5. The majority of the packets are locally consumed. Reference [16] reported that 72% of the traffic was intranetwork.
6. The raw channel bit error rates are practically negligible, being of the order of 1 in 10^{11} to 10^{12}.[26,27] The observed error statistics in an operational environment is small;[23,17] reference (16) reported an error rate of 1 in $2*10^6$ packets.
7. Even in a busy large scale Ethernet, the probability of packet collision is small. Reference (17) reported negligible collisions up to 1,000 users. With 2,000 users, the collision rate was about one per successful transmission. The ninetieth percentile waiting time was 400 usec with about 1,400 users. The mean waiting time was around 100 usec. Thus the theoretical possibility of an unbounded packet delay is a non-problem with current operational environments.

3 Factors Influencing Protocol Performance

The performance of a transport level protocol depends on the following factors;

1. the control structure of the protocol[1]
2. the flow control mechanism if it is a stream protocol
3. The volume of control packets generated by the protocol[4]
4. The way the protocol is implemented and its relationship to the host operating system[4]
5. The overhead induced by the host operating system and the structure of the network interface.[4]

These factors were considered in the performance evaluation of TCP/IP in our LAN system.

3.1 Control structure

The control structure of a transport protocol deals with the packet format, retransmission and timeout policies, client and network level interfaces, packet fragmentation and reassembly, congestion control, peer addressing, etc. TCP/IP[2,3] was designed to provide internet packet transport over lossy subnets. The services provided include internet address handling, routing, internet congestion control and error reporting, multiple checksums (one at the TCP level and the other at the IP level), segment fragmentation and reassembly, datagram self-destruction mechanisms, and service level options to clients. The large, byte level sequence numbers used in TCP (thirty-two bits) incurs considerable processing overhead[28] but is justified in the context of an LHN on the grounds that a large recycling interval allows easy identification of out-of-order and duplicate segments and that byte level sequence numbers facilitate the fragmentation of segments at the local IP layer and the intervening gateways, and the reassembly at the remote TCP entity. These service features are seldom used (in fact some of them are irrelevant) in a single LAN environment. They nevertheless consume considerable protocol processing time[1] which is a critical resource in a LAN and constitutes a performance bottleneck in a LAN environment.[28] However, since it is part of the protocol itself, one cannot proceed to improve performance by modifying the control structure without compromising the protocol specifications. See section 8.

3.2 Flow control

The TCP specifications do not specify the type of flow control to be implemented: only recommendations are made. (See [2] for details.) The TCP implementation in 4.2 BSD UNIX has a simple flow control scheme. A flow control parameter α is defined which is the ratio of the number of buffers released (NP) since the last window update and the total number of available buffers (NB). That is,

$$\alpha = \text{NP/NB}. \tag{1}$$

When α exceeds a preset value α_c, an acknowledgement packet containing the current window size is sent to the sending peer. α_c was originally set at 0.35, which means that when the number of packets consumed exceeds 35% of the total buffer space, a window update becomes due. Sender and receiver buffers were set at 2Kbytes each (see figure 1). It was not clear whether the flow control and the buffer sizes were adequate. We believe, however that lax flow control will result in an uneven flow of packets because of the frequent choking of the protocol. On the other hand, a rigorous flow control scheme would generate excessive control traffic

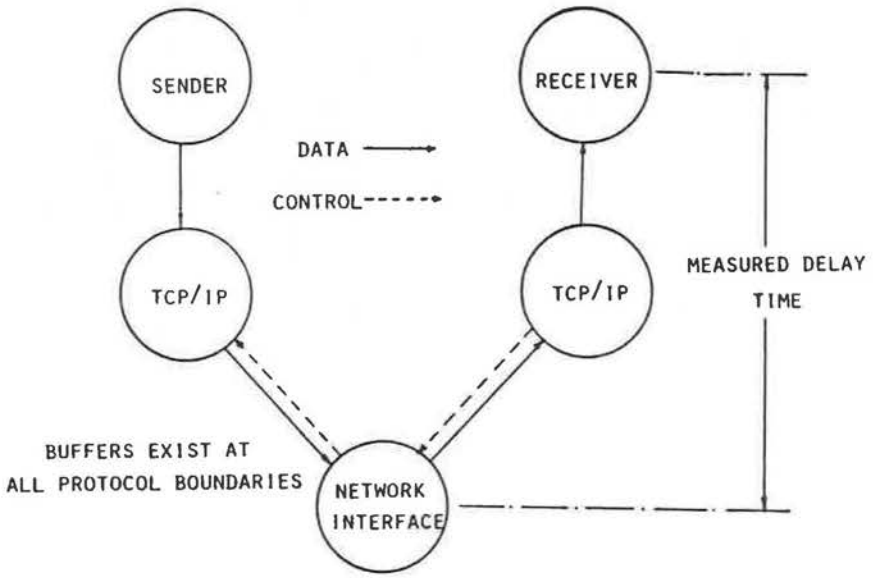

FIG. 1. Implementation model of TCP/IP in BSD 4.2 UNIX

which is wasteful of network resources. Thus it was felt that some detailed measurements are necessary in this respect.

3.3 Control traffic

The third factor, the volume of control traffic, is closely related to the flow control policy. If the protocol treats window updates and packet acknowledgements separately, then this factor should be studied separately from flow control. In TCP, however, since the acknowledgement and the window update go in the same packet, study of flow control sheds light on this factor also.

3.4 How the protocol was implemented

The fourth factor is the logical relationship between the protocol and the host operating system. There are a variety of choices available to the implementor.[4]

1. The protocol runs as a user process.
2. The protocol runs as a privileged process with direct mapping into the kernel address space.[20]
3. The protocol runs completely within the kernel of the host operating system.
4. The protocol runs on a separate front-end processor.

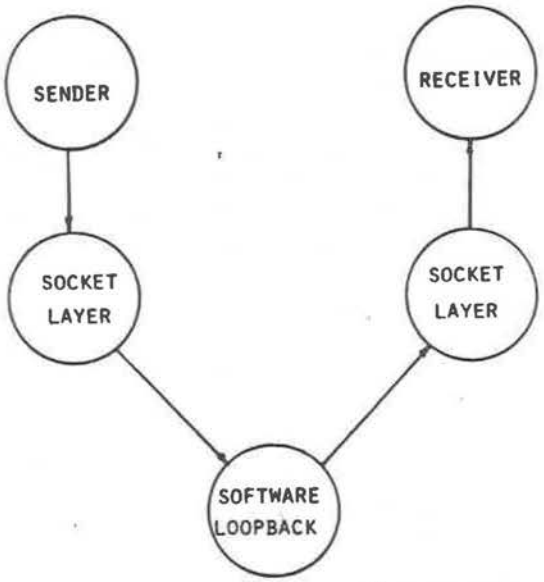

FIG. 2a. Software loop back mode IPC (without protocol support)

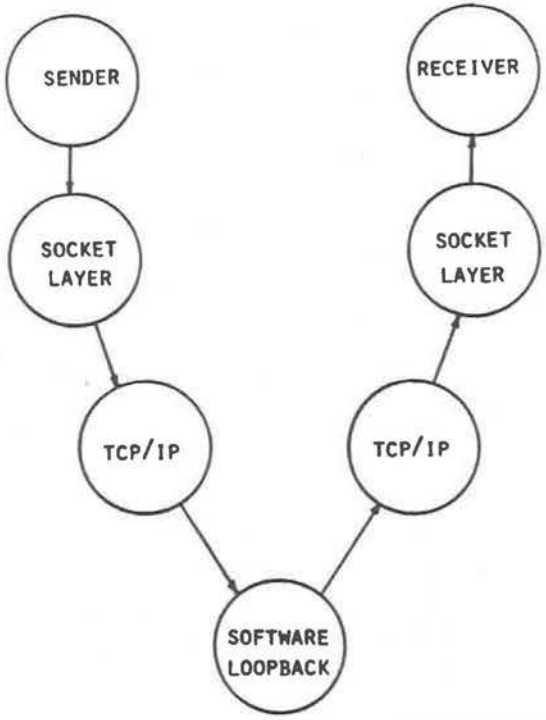

FIG. 2b. Software loop back mode IPC (with protocol support)

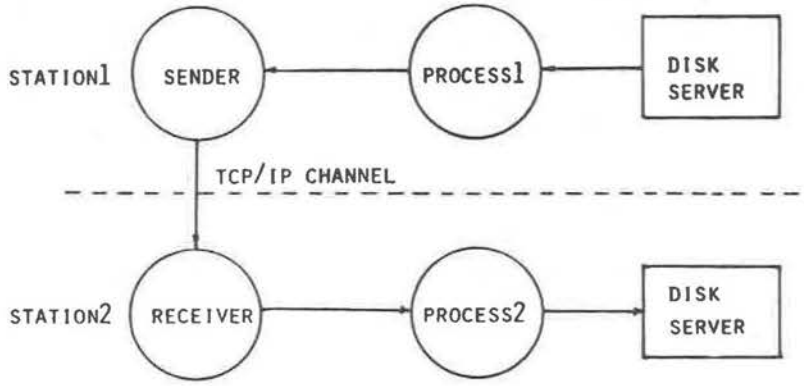

FIG. 2c. IPC setup with disk i/o included

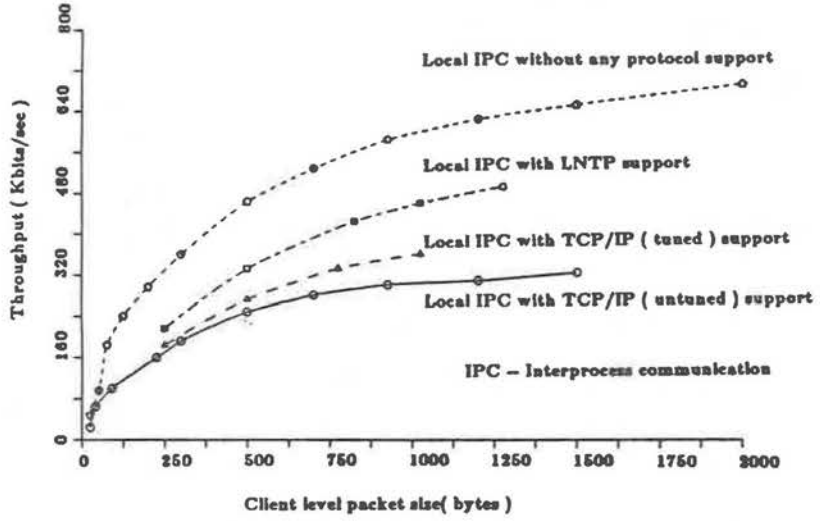

FIG. 3. Software loop back test results

5. Some of the functional layers of the protocol run inside the kernel and the rest as user level processes.

It is beyond the scope of this paper to discuss the relative merits of each. We shall mention however that 3 and 4 give the best performance. The interested reader is referred to [4] for a detailed discussion on this subject. In 4.2 BSD UNIX, TCP/IP has been implemented in the kernel, which means that all the kernel-provided services, such as timers, device control, and memory buffers, are readily available to the protocol. This is made possible because of the availability of a large address space in the

host machines (SUN and VAX) and of the large memory size. Thus it was felt that not much could be done to improve performance in this area, except of course to offload the protocol into a front-end processor (some of the commercially available network interfaces provide large buffers and software support for this offloading).

3.5 System overhead and the network interface structure

3.5.1 Operating system overhead

Operating system overhead is another factor that influences the protocol performance. For example, how data are passed among the various processes, the host-scheduling philosophy, the queue service disciplines, support of non-blocking i/o and context switching all affect the performance in some way. In 4.2 BSD UNIX, IPC-related calls are non-blocking; that is, the sender need not wait for the packet to be delivered to the receiver and acknowledged. Buffers are managed in units of 112 bytes each. This leads to a fragmented representation of data, and all packet processing is done on a chained list of buffers, which entails considerable overhead. Our measurement of 4.2 BSD UNIX running on the SUN workstation showed this overhead to be around 45% to 50%. The software loopback measurement results described in [1] performed on a PDP-11/70 running a modified version of UNIX indicate that 70% of the processing time is consumed by the system overhead in the form of system calls, context switching, etc.

3.5.2 Network interface structure

The network interface unit (NIU) used to access the communication channel also strongly influences the performance of a transport connection. Our systems (both VAX and SUN) use Ethernet interfaces from 3COM Corporation which do not support true DMA. The interface has a pool of 2Kbyte buffers,[10,11] each of which can independently be controlled either by the host or the NIU at any time. (The number of buffers and their organization, however, are different in the two systems.) After a packet is processed by the protocol, the host copies it from its memory to the NIU buffer. The NIU then transmits the entire packet, and interrupts the host. Similarly an arriving packet interrupts the host, which then copies it into its memory. This structure is in contrast to that of a true DMA device which either has a dual-ported local buffer or can directly access the host memory across the system bus. The extra copy needed in our system can be circumvented if the packet is copied from the application data space directly into the NIU buffer. But this requires the packet routing decision to be made at the client layer which is above the transport level. This is inconsistent with the OSI reference model[21] which places the routing decision at the network layer.

Not wishing to modify the host operating system or change the NIU, we decided to probe in detail the protocol related aspects, namely the flow control scheme, the number of buffers and the control traffic. A performance study of TCP in the context of *long-haul networks* has been performed at University College, London.[5] In this study the focus was essentially on retransmission strategies and segment sizes. Flow control related aspects were studied to some extent, but the protocol was never considered a bottleneck.

4. MEASUREMENT EXPERIMENTS

4.1 Experiment 1

The first experiment was designed to measure the performance of the existing TCP implementation in some quantitative terms. In this experiment the client and the server processes reside on the same machine where a steady stream of data flows from the client to the server (see figure 2a). At the network interface level, the system provides a software loopback interface which mimics a physical network and echoes back the data to the receiver. In this experiment, data were transferred with and without protocol support. See figures 2a and 2b. From the results (figure 3) several observations can be made.

1. The performance without protocol support forms an upperbound on the performance of any protocol implementation in our LAN system.
2. The TCP performance figures were only half of the upperbound (350Kb/sec as against 700Kb/sec).
3. The packet level processing that is data independent, such as header processing, constitutes a sizeable fraction of the total packet processing time. This can be seen by the saturating behaviour of the throughput curves as packet size is increased. The throughput rate R (bits per second) is approximated by the simple relation.

$$R = 8 * n/[2 * (\text{TSYS} + \text{THDR} + n.\text{TPKT})], \qquad (2)$$

where

n = Packet size in bytes
TSYS = System call handling time (= 1.5 msec in 4.2 BSD UNIX on the SUN workstation)
THDR = Header processing time

TPKT = Effective packet processing time per byte of useful data (it includes the processing time for control traffic, data copying, checksum evaluation, etc.).

When n is small, the overhead per byte of data transferred is very large, and R varies considerably with n. For large n, n.TPKT $\gg$ (TSYS + THDR) and R tends to be a constant.

From observation 3 we conclude that the packet size should be as large as possible. We recommend that a reasonable packet size at the client level should be 1,024 bytes, which is a frequently used unit of disk storage.

4.2 Experiment 2

Instead of using the loopback mode, one of the processes was moved to the VAX11/750 running the same version of 4.2 BSD UNIX (see figure 1). Using TCP/IP, large volumes of data were transferred in both directions, first from VAX to SUN, then from SUN to VAX. The throughput rate in the VAX-SUN transfer was about 400kb/sec for segments of 1,024 bytes whereas the reverse case yielded only about 340kb/sec. Even though a twofold increase in throughput rate with respect to the loopback mode testing was not expected (because in the actual transfer, servicing the network interface constitutes about 20% of the packet processing load), we did not expect the performance to be only marginally better than that of the loopback mode. We felt that inefficiency existed in the protocol either in the form of excessive control traffic or an uneven flow of packets or both. So the next step was to monitor the packet level activities within the protocol.

4.3 Experiment 3

The aim of the experiment was to monitor the packet flow distribution, since we strongly believe that any misbehaviour on the part of the protocol will show up as uneven packet flow (the converse is not necessarily true). The distribution of packet sizes and packet interarrival times were monitored at the protocol boundary between TCP/IP and the Ethernet. The results we obtained revealed the following:

1. Data were heavily fragmented within the protocol (not the IP level fragmentation) even though the messages were all sized well below the Ethernet specified maximum of 1,500 bytes (see figure 4b).
2. The packet flow was very uneven (and so was the control traffic) as seen by the widely spread-out packet interarrival time distribution curves (see figure 4a).
3. There was a heavy volume of control traffic, about one control packet for every data packet.

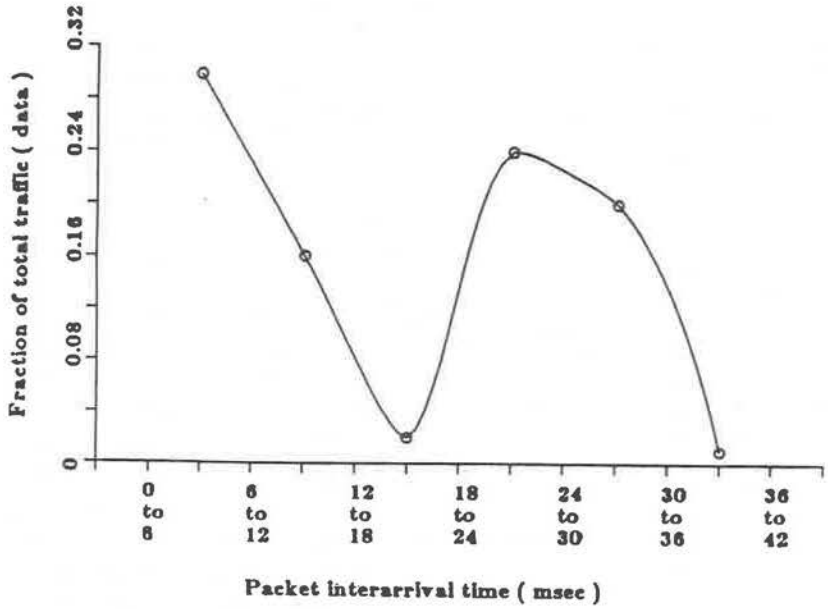

FIG. 4a.1. Flow distribution of data packets (untuned TCP)

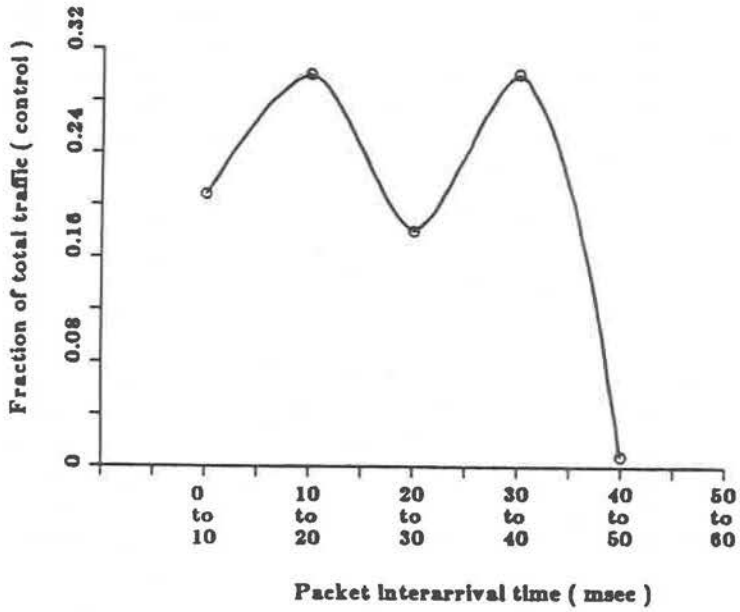

FIG. 4a.2. Flow distribution of control packets (untuned TCP)

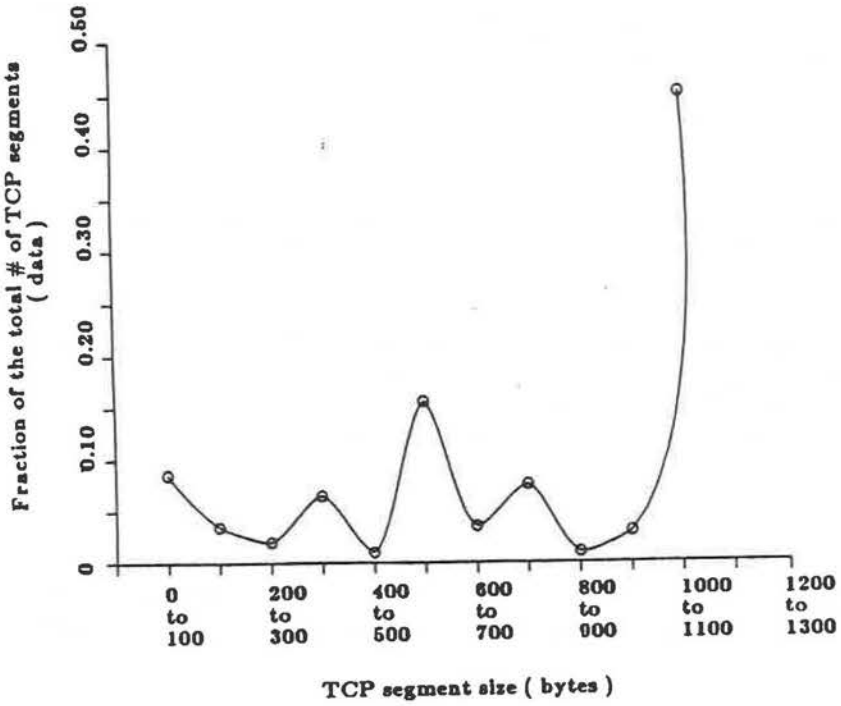

FIG. 4b. Size distribution of data packets (untuned TCP)

We concluded that the 2K buffer space provided in TCP for each of the sender and the receiver was too small, resulting in the exchange of small and varying window sizes. Thus the sending of peer blocks frequently for want of window space. Even if the sender has a window available, it is often smaller than the client specified size. So the TCP fragments data to a size equal to that of the available window before sending. This explains the heavy fragmentation of data. Since buffers cannot be released without a positive acknowledgment from the receiver, the sender tends to choke very frequently. This happened about 925 times when 10,000 packets were transferred from the SUN to VAX. For the reverse traffic, the faster sender (VAX) was getting blocked at least 3,000 times for the same amount of traffic. The time average of the window space was only about 1,000 bytes, rather small for a stream protocol in LANS where the normal unit of flow is 1,024 bytes.

4.4 Environmental interferences

The environmental factors that could affect the measurements are the interference due to unrelated work loads presented by other users, the disk i/o initiated by the end applications handling the streams, other regularly activated daemon processes, and the mail traffic arriving over the network. The interference due to daemon processes is characterized by short bursts of computation and exchange of short packets separated by large intervals of inactivity; hence its effect is negligible. The message traffic was light in our system and arrived in random (of the order of one to two messages per minute). This interference too was found to be negligible.

The question of whether to include disk i/o as part of the measurement process was carefully considered. The SUN workstation uses a FUJITSU disk drive with an average service time (access and data transfer) of about 30 msecs. The disk drive on the VAX has comparable specifications. Since the network i/o is of the order of 15 msec per packet, a sufficient overlap of disk and network i/o can make the effect of disk i/o invisible. An asynchronous process may be created to fetch mutiple disk blocks in a single request and pass them to the application process for network transfer (see figure 2c). It was found that with single disk requests of four disk blocks, the application process rarely gets blocked for want of data. However, there is the associated overhead in transferring data between the processes, context switchings, and disk driver processing.

The interference due to unrelated work loads on the same workstation is completely random in nature and non-reproducible. It is very difficult to incorporate its effect in the study. However, as workstations are typically used by no more than one user at a time, the exclusion of this interference from the measurements might not be unrealistic.

Regarding unrelated load on the Ethernet from other stations, measurement results from large Ethernet installations elsewhere,[16,17] as discussed in section 2.4, showed no collision for up to about 1,000 users. This means that except for extremely heavy workload on the Ethernet bus, a station can access the bus without collision. Thus, within limits, measurements across a pair of stations are largely independent of other network activities. This fact was verified on our LAN system by the following experiments. Experiment 2 (VAX-11/750 to SUN file transfer) was repeated with additional file transfer traffic on the Ethernet from a second VAX-11/750 to a VAX-11/780. The added traffic was created first by two pairs of concurrent processes and again by sixteen pairs transferring files simultaneously. Each experiment was repeated six times. During the experiments electronic mail messages were coming in and processed by

TABLE I

Throughput rate†	Quiet network	Loaded* network	Loaded** network
Mean (Kbps)	400	400	398.5
Standard deviation (Kbps)	2	6	3.5

*two additional concurrent file transfers
**sixteen additional concurrent file transfers
†VAX-11/750 to SUN file transfer

the first VAX-11/750. The results of the experiments are summarized in table 1. The maximum difference in mean throughput rate was under 0.4%.

It was therefore decided to study the performance of the TCP/IP without any disk i/o and unrelated workloads. One should note, then, that the figures so obtained would provide upper bounds on the performance of the virtual channel between the transport level entities.

5. Tuning Therapies

Having located the bottlenecks, our next step was systematically to tune the protocol parameters, namely the number of buffers and the flow control parameters α_c.

5.1 Optimal buffer size

The net throughput rate and the mean packet delay time were taken as the primary performance indices. To make a systematic study of the effect of the buffer size on the performance indices, a simple deterministic model was formulated which allows the throughput rate to be estimated in terms of the buffer size and the flow control parameter.

5.1.1 Throughput model

Consider the queueing model of figure 1. Since throughput is limited by the slowest server in the chain, the model essentially deals with parameters with respect to the slowest server.

Let TDAT = Mean service time per data packet
TACK = Mean service time for an acknowledgement packet

Mean effective service time per data packet = TDAT + TACK/NP,
where NP (< NB) is related to the flow control parameter.

Normally, NP is chosen to be three to four buffers less than NB as specified by the flow control model of section 5.2.1. It is the number of packets received and consumed since the last window update. So the maximum throughput rate achievable assuming a steady traffic is

$$R = 1/(\text{TDAT} + \text{TACK/NP}) \quad \text{for a fast sender}$$
$$= 1/(\text{TDAT} + \text{TSCH} + \text{TACK/NP}) \quad \text{for a slow sender.} \qquad (3)$$

TSCH is the average time a packet waits in the receiver's input queue before the protocol layer is notified of its arrival. This is a system related parameter and is heavily influenced by the host operating system's scheduling strategy. For example, with VAX-11/750 as the receiver (SUN being the sender), TSCH = 4 to 5 msec. In the case of a fast sender, TSCH = 0 in our system, since the slow receiver always finds a packet in its input queue waiting to be processed, and the protocol process never relinquishes the CPU until its input queue is empty. This asymmetry shows up as reduced throughput rate for the SUN-VAX transfer (about 20%) compared with the VAX-SUN transfer.

Reference [6] describes a queueing-model based theoretical analysis to compute the upper bounds on throughput rates when buffers constitute the bottleneck in the system. The model does not take into account the variation in control traffic that might arise as the buffer size is varied. We have extended and generalized the model to yield the upperbound on throughput rate RU.

$$\text{RU} = \text{NB}/[(\text{TDAT} + \text{TACK/NP})(\text{KS} + \text{KR}) + \text{TDN} + \text{TAN}], \qquad (4)$$

where 1 < = NP < = NB, and TDN and TAN are the network delay times for the data and acknowledgement packets respectively. KS and KR are the speed factors for the sender and the receiver respectively (see appendix A). Thus as NB increases, the optimal value for NP also increases, and RU has an increasing slope. Beyond NB = 3, receiver processing becomes the bottleneck, limiting the throughput rate. The observed throughput rates are well within these computed bounds. They also match well with equation (3) (see figures 7a and 7b).

5.1.2 Tuning experiment 1

An experiment was designed to monitor the overall throughput rates and the mean packet delay times as a function of the buffer size. The flow control parameter in each case was set as dictated by the model described in section 5.2.1. The sender was the faster VAX with the SUN workstation the receiver. The waiting times in the receiver queues were monitored. Each packet was timestamped on entry into the TCP/IP input queue from the network interface. An updated timestamp was affixed on each packet

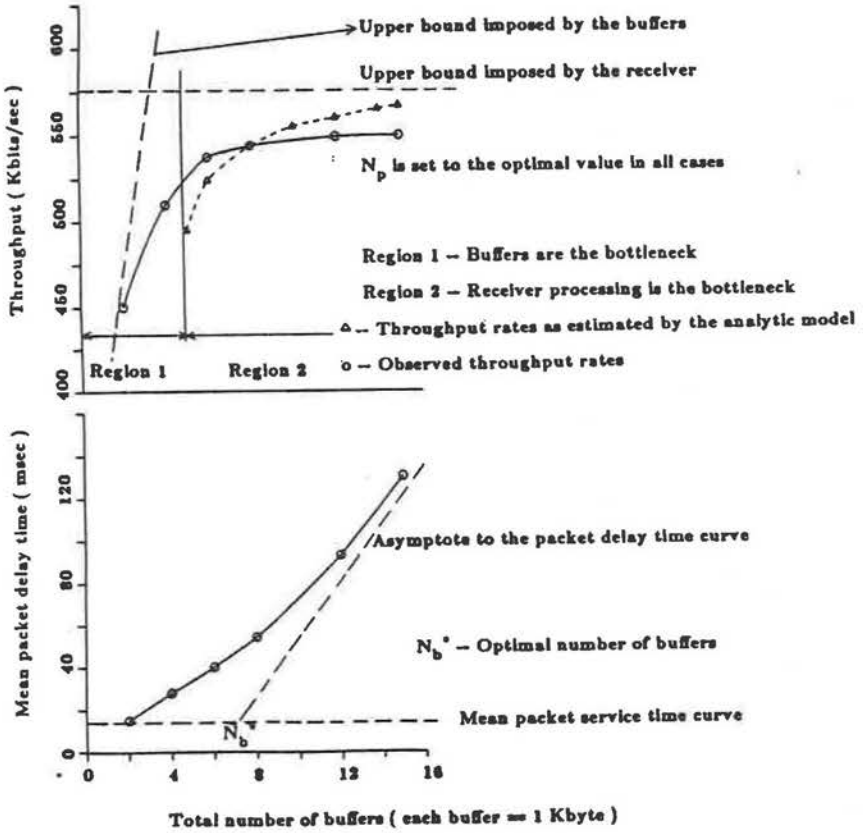

FIG. 5. Performance variation with respect to the number of buffers

on exit from the input queue of the socket layer. The delay time distribution was also observed. It was found that the delay time of each packet had a lower bound that is much larger than the mean packet service time. This was a sensitive function of NP and NB. In the 15k buffer case, the mean packet delay time was about 130 msec whereas in the 8K case, the delay was about 55 msec, with NP set at the optimal value in both cases. The delay time was also heavily dependent on the flow control parameter. The variation of the throughput rates and the packet delay times with respect to the buffer size is as given in figure 5. The observed throughput results match quite well with those obtained using equation 3 with the flow control parameter (and hence NP) set at the optimal value as described in section 5.2.1. One can see that the throughput rate saturates when the buffer size reaches 8Kbytes, whereas the packet delay times increases

monotonically. This behaviour is analogous to that of a multiprogramming system with respect to the degree of multiprogramming.[22]

With this experimentally observed throughput rate and delay time behaviour, we applied Kleinrock's criterion[7] with respect to the asymptotic behaviour of the system. The point of intersection of the asymptote to the delay time curve with the horizontal line representing the packet service time at the receiver fixes the optimal number of buffers that keeps throughput and delay time within acceptable limits. This was found to be between 6 to 8Kbytes. The throughput rate at this point is about 530Kb/sec, and the packet delay time is around 40 msec (figure 5).

5.2.1 Flow control model

Having fixed the optimal buffer size, the next stage was to determine the optimal flow control parameter value that maintains an even flow of packets with a minimal amount of control traffic. A simple deterministic model was formulated for this purpose (see appendix A). The model specifies the minimum amount of cushion needed at the receiver that satisfies the above conditions. The number of buffers released which generates an acknowledgement packet is given by the inequality.

$$NP < = NB - [KR.NW/(KR + KN + KS)] - [(KR + KN + KS)/KS][TACK/TDAT] - [KS + KN]/KR. \quad (5)$$

To minimize the control traffic NP should be set equal to the expression on the right hand side of equation (5). This value of NP is easily mapped onto the TCP's flow control parameter by the simple relation (1) which is rewritten here as

$$\alpha = NP/NB.$$

5.2.2 Tuning experiment 2

Experiments were conducted to monitor the throughput rate, packet delay time, control traffic, and the time average window size as a function of the flow control parameter α. The experiments were carried out with NB = 8k and 15k bytes. The optimal values of α obtained from the experimental results match quite well with the value proposed by the model. The results are as shown in figures 7a and 7b. The variation of the time average window size with respect to α is quite linear. Notice that a small α keeps the packet flow even and steady at the expense of increased control traffic. For $\alpha > \alpha_c$, the protocol gets blocked for want of window space, but the control traffic is reduced. At $\alpha = \alpha_c$, the control traffic is just enough to maintain a steady flow without choking. It is observed that the performance degradation due to protocol blocking is more severe than

that due to increased control traffic. This implies that an α slightly less than α_c will be a safe region of operation. As seen in figure 6, the flow distribution shows sharp peaks indicating an even and unchoked flow. The control traffic is high but the protocol never blocks more than necessary.

6. Guidelines to Protocol Design and Evaluation

Based on the analysis of the measured data as well as empirical data obtained elsewhere (see section 2.4), certain guidelines concerning the design and tuning of TCP-like protocols are presented below:

1. The number of retransmissions at the TCP layer were practically zero in our small LAN system. Even in large LANs, the probability of retransmission is very small.[16,17,23] This means that a LAN transport protocol need not place much emphasis on retransmission-based error recovery schemes. Similarly, run-time connection management like connection resets, forcible shutdown of the connections with zero in our observations. Thus a LAN transport protocol requires only a simple connection management scheme (see also[8]).
2. An end-to-end application seldom loads the network beyond a small fraction of the network capacity. In our measurements, a streaming application presents a maximum load of 6% while an interactive application (send-receive-reply) presents a maximum load of about 0.35% to 0.5% of the Ethernet capacity of 10 Mb/sec. Spector[15] reported that the maximum interactive load presented by a single Alto machine on a 2.94 Mb/sec Ethernet is about 1.8%. This means that the subnet is almost never the bottleneck and the network delay times are small compared to the protocol processing times. So flow control should be strong and adaptive enough to accommodate the different speeds of the host machines and the varying work loads so that packets do not get discarded at the receiver for want of resources. This requirement is not as important for LHNs where the transmission speeds are typically several orders of magnitude slower; thus the probability of a fast sender swamping a slow receiver is much less than that in a LAN environment.
3. It is not efficient to send packets in small sizes.[8,15]
4. Checksumming and data copying operations should be kept to a minimum.[4]
5. The packet flow distribution can provide a conclusive pointer as to whether the flow control policy is a serious bottleneck. An inefficient flow control due either to the lack of buffers or delayed acknowledge-

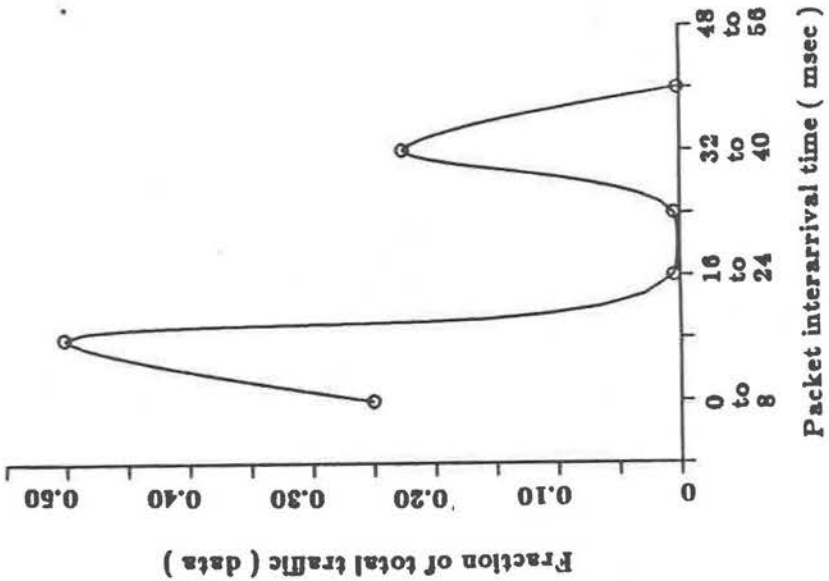

FIG. 6a. Flow distribution of data packets (tuned TCP)

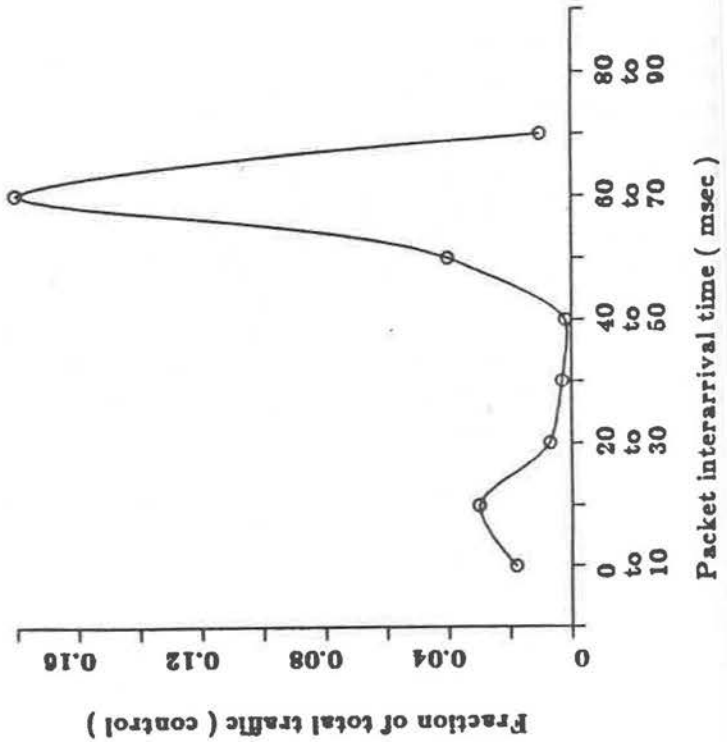

FIG. 6b. Flow distribution of control packets (tuned TCP)

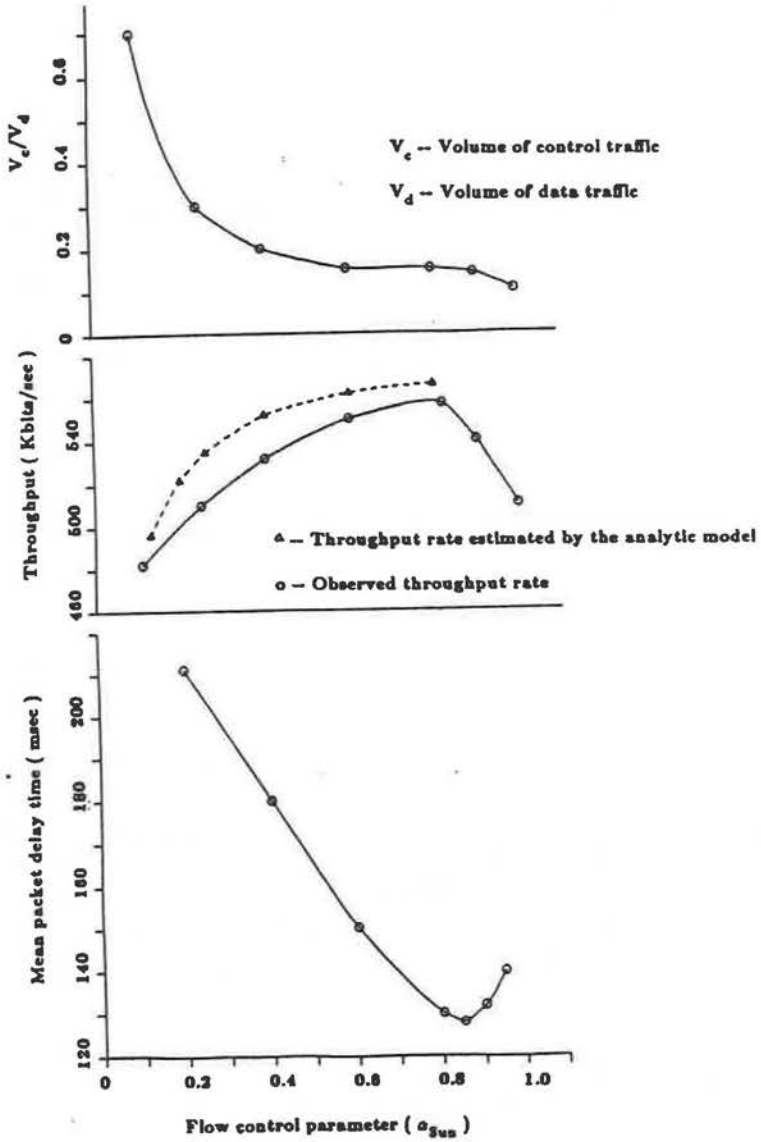

FIG. 7a. Effect of flow control parameter on performance (N_b = 15 Kbytes)

ments results in an uneven flow of packets that shows up as a wide distribution (i.e., without sharp peaks).

6. Control traffic marginally higher than the required minimum keeps the flow steady with the throughput and packet delays close to the optimal values. It is not desirable to allow the protocol to get blocked more than necessary, since this is more detrimental than allowing a slight increase in control traffic.
7. Data should never be fragmented. This can be achieved by avoiding extremely small windows and allowing updates that are integral multiples of the frequently used packet sizes.
8. Protocol performance is highly sensitive to the relative speeds of the

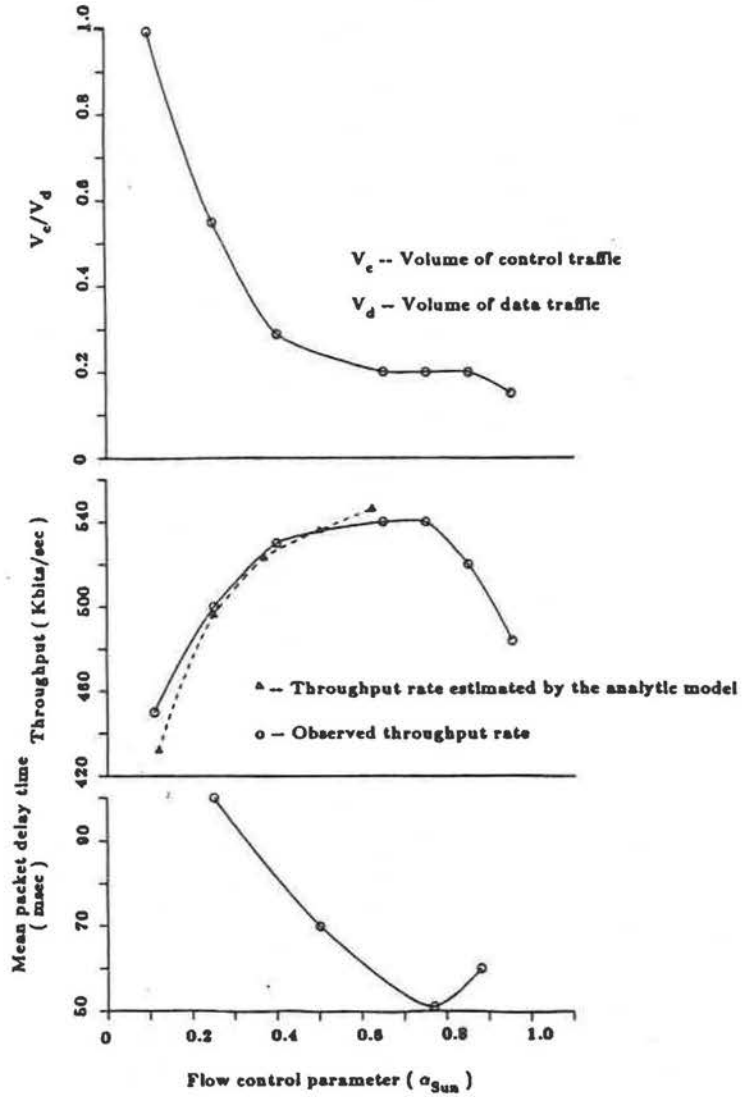

FIG. 7b. Effect of flow control parameter on performance (N_b = 8 Kbytes)

host machines, the resources available per connection, the characteristics of the hardware network interface, and the system environment. The dependency of the protocol parameters on these factors should be carefully considered in protocol design and implementation.

7. Comparison with the Performance of Some Other Protocols

We briefly compare our results on TCP/IP performance with other implementations of TCP/IP as well as the DECnet protocol. This comparison is based on the informal measurement results that are distributed across the TCP/IP mailing list in the ARPANET mailing system.[13]

7.1 DECnet performance over Ethernet

DEC's publication[12] provides data on DECnet V3.1 performance over Ethernet under VMS. The maximum task-to-task transfer rate between two VAX-11/780's with an internal buffer of 576 bytes and user buffer of 4 Kbytes is 800 Kb/sec while that for VAX-11/750's is 600 Kb/sec; the file transfer rates (presumably including disk server processing) are 420 Kb/sec and 350 Kb/sec, respectively. With link optimization, that is, an increased internal buffer size of 1,498 bytes, the throughput rates are 1.3 Mb/sec and 1.2 Mb/sec, respectively, for task-to-task transfer, and 500 Kb/sec and 390 Kb/sec for file transfers.

7.2 Other TCP/IP implementations

Both UNIX 4.2 BSD TCP/IP and VSM/EUNICE/TCP/IP implemented on the Ungermann-Bass broadband network yield a file transfer rate of about 200 Kb/sec. TEKTRONIX TCP/IP under VMS gives a file transfer rate of 140 Kb/sec between a VAX-11/780 and a VAX-11/750 over an Ethernet with Interlan controller using blocking VMS i/o calls as against 280 Kb/sec for VMS/DECnet. No results on possible performance improvement are available with non-blocking i/o calls. The corresponding figure for TCP/IP between two VAX-11/780s is 200 Kb/sec. Similarly the VMS/TCP/IP to UNIX 4.2 BSD TCP/IP file transfer rate (again between VAX-11/780's) is 200 Kb/sec with task-to-task communication at 400 Kb/sec.

From the above informal benchmarks, it would appear that the DECnet protocol outperforms TCP/IP in a LAN environment. However, another study by Jacobson et al. at Lawrence Livermoore Labs (again circulated in the ARPANET mails[13]) claims that both the UNIX TCP/IP and VSM TCP/IP outperform VMS/DECnet, with the UNIX/TCP/IP performing better than its VMS counterpart; no quantitative results are given. Thus, due to the informal nature of these benchmarks and the lack of comparative studies in identical environments, it is difficult to draw firm conclusions as to the performance of TCP/IP relative to other protocols or even with implementations of TCP/IP under different systems. Our study has mainly concentrated on the tuning aspects of TCP/IP under 4.2 BSD UNIX; nevertheless, our measurement data on TCP/IP performance are in line with the above informal benchmark results.

8. A New Transport Level Protocol for LANs

All the tuning procedures mentioned in the earlier sections are based on flow control and buffer-related aspects only. Nothing was done to the control structure of the protocol itself, since this would lead to violations of the protocol specifications. To improve performance further we came up with a simple protocol LNTP (Local Network Transport protocol) that would provide the same functional service as TCP/IP for the local net traffic. The observed traffic characteristics in other large scale LAN installations, as discussed in section 2.4, were taken into account in the protocol design. The design emphasizes

1. a simplified control structure,
2. a clean flow control scheme with minimal interactions between sender and receiver.

The protocol is asymmetric in structure, so that the processing at the sender and receiver is highly localized in functionality. Since the protocol is to support a bidirectional stream interface to the client layer, a complete implementation has two complementary instances of the protocol to provide such an abstraction. This keeps the protocol implementation much cleaner and elegant. The main features of the new protocol are as follows:

1. It provides a virtual circuit service on top of a datagram service.
2. It encapsulates only the very few protocol functions that are essential in a single LAN environment, leaving out the unnecessary complexities typical of a LHN protocol. Specifically, the internet address generation, routing, packet fragmentation and reassembly, different service level options provided to clients, congestion control and error reporting, multiple checksums and packet self-destruction mechanisms have been eliminated. This has resulted in considerable redution in the protocol overhead apart from the added advantage of a reduced packet header size (from 40 bytes to 10 bytes). Furthermore, together with the clean flow control mechanism, this has also resulted in a very small set of control packet types.

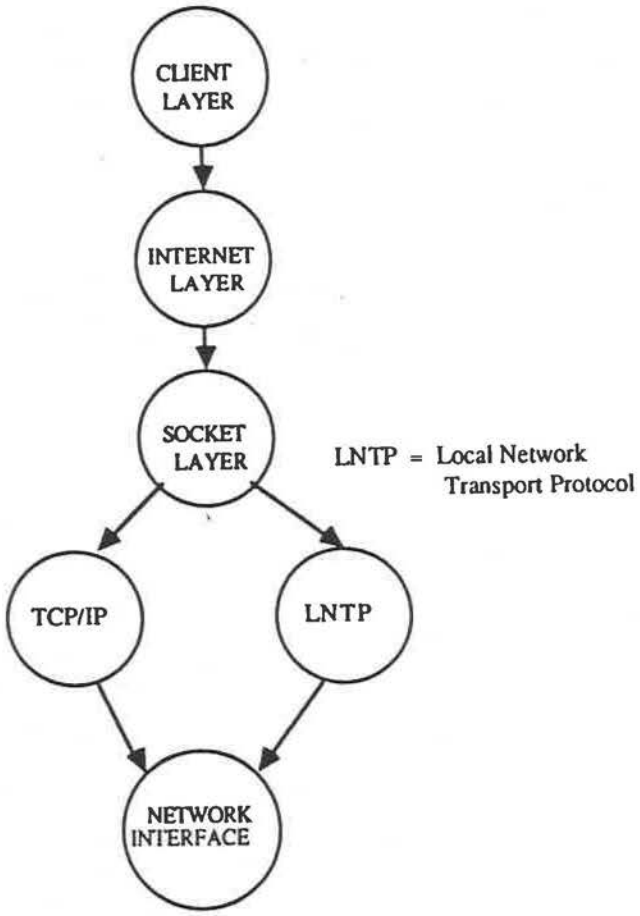

FIG. 8. Implementation model of LNTP

3. A deferred flow control scheme that provides a maximum degree of parallelism between sender and receiver.
4. Small, packet level sequence numbers (as opposed to large, byte level sequence numbers used by TCP) for simplicity in processing.

A brief description of LNTP is presented in appendix B. Details can be found in [28]. A preliminary implementation of the protocol in the 4.2 BSD UNIX kernel has been made at the same functional level as TCP/IP (see figure 8). It has been tested in software loopback mode to get a comparative view of the protocol in terms of performance. The protocol gives a good improvement in throughput rate (around 450kb/sec compared with 360kb/sec for the TCP/IP under identical set of optimized

protocol parameters, see figure 3). Actual transfer of data with other hosts in our LAN system could not be done at this time, because the vendor support for the Ethernet is provided only for TCP/IP packets. Since we did not have the source code for the Ethernet driver at the time of writing, we could not provide the Ethernet support for our new protocol. Only after carrying out real transfer experiments can the contribution to performance improvement due to the modified flow control, control traffic, and the simple structure be conclusively determined.

9. Plans for the Future

Ethernet support for the new protocol will be provided. The protocol design and implementation are more or less complete. The performance of distributed applications supported by this protocol will be studied. Once the protocol is proved reliable and robust, all the distributed applications running under 4.2 BSD UNIX in our LAN system (currently supported by TCP/IP) will be supported by this new protocol. Though non-standard, it can provide an efficient communication support within the LAN system. When it is necessary to communicate with other systems, full TCP/IP can be implemented at the gateway.

10. Conclusions

Measurement experiments relating to the performance of the 4.2 BSD UNIX implementation of TCP/IP supporting the transfer of large files between hosts of different speeds connected by a 10 Mb/s Ethernet were described. A methodology was proposed which may detect the bottleneck in the protocol. Models were presented which allow the determination of the optimal buffer size and the flow control parameter. We also described a new transport protocol for local area networks with much simpler control structure which provides the same functional service (excluding the internet packet transport) as TCP/IP in 4.2 BSD UNIX with significant increase in network throughput. Finally, summing up what we have learned, a set of guidelines to protocol design and evaluation were presented.

It is hoped that our experience in measuring and tuning TCP will be of use to others running the same system as well as to designers and implementers of protocols for local area networks.

Appendix A: A Deterministic Model for Flow Control in TCP

The following analysis assumes that the input queue is never empty; that is, there are always some data to send.

If we let

TDAT = Mean packet processing time including interfacing to the client layer
TACK = Mean processing time on an acknowledgement packet,

then TDAT > TACK, because

1. the data-dependent processing, such as copying, checksumming, and manipulation of memory buffers, need not be performed for the acknowledgement packet;
2. the acknowledgement packet does not reach the client interface.

TDAT and TACK are specified with respect to "reference machine." Let

NW = mean number of packets in the system
KS = speed fator for the sending machine
KR = speed factor for the receiving machine
KN = speed factor for the Ethernet layer.

For the reference machine, the speed factor is 1.0. Thus for a machine with a speed factor K, K.TDAT and K.TACK are the protocol-processing times for the data and acknowledgement packets, respectively. Let

WS = size of the window at the most recent update
NP = number of packets cleared from the receiver socket buffer since the last window update

Two cases are considered.

Case 1. Sender slower than receiver (KS > KR)

Example, SUN to VAX1/750

When the NPth packet has been received, the window as seen by the sender is given by

$$W = WS - NP - (KR + KN)/KS. \quad (A1)$$

The last term represents the number of packets that are already in the network.

Mean proportion of traffic at the input queue of the receiver
$$= KR/(KS + KR + KN). \quad (A2)$$

Since the acknowledgement packet has to traverse the downstream path against the incoming traffic, which has higher priority over the outgoing traffic, the time for the acknowledgement packet to reach the input queue of the sender is given by

$$T' = (KR/[KS + KR + KN]).NW.KR.TDAT + (KR + KN).TACK. \quad (A3)$$

If the sender gets the update before or just as its window closes, then the flow will be continuous, and there will be no choking. This condition can be mathematically expressed as

$$T' <= [WS - NP - (KR + KN)/KS].KS.TDAT.$$

From (A3),

$$NP <= \{[WS - (KR + KN)/KS)] - KR**2.NW/[(KS + KR + KN)\cdot KS] - (KR + KN).TACK/(KS.TDAT)\}. \quad (A4)$$

The control traffic is minimized if the right- and left-hand sides are equal.

Case 2. Sender faster than receiver, that is, KS < KR

Example, VAX11/750 to SUN

The mean proportion of packets at the receiver input queue
$$= KR/(KS + KR + KN). \quad (A5)$$

An acknowledgement packet is sent by the receiver to the sender after the receipt of NP packets since the last window update. The time taken for this acknowledgement packet to reach the sender to enable transmission of a subsequent data packet and for this data packet to arrive at the receiver should be just sufficient for the receiver to complete processing the remaining (WS − NP) packets. This will minimize the control traffic without idling the slow receiver. Mathematically, the above time is given by

$$T'' = (KR + KN + KS).TACK + [KR/(KS + KN + KR)].NW.KR.TDAT + (KS + KN).TDAT. \quad (A6)$$

Since

$$T'' <= (WS - NP).KR.TDAT$$

therefore

$$NP <= \{WS - [KR.NW/(KR + KN + KS)] - [(KR + KN + KS).TACK/(KR.TDAT)] - (KS + KN)/KR\}. \quad (A7)$$

Equations (A4) and (A7) provide some general guidelines for determining the value of NP which can easily be mapped onto the TCP control parameter. A numerical example is presented below.

The parameters for SUN and VAX11/750 are (the SUN is taken as the reference machine)

$$K(SUN) = 1.0,\ K(VAX) = 0.9$$
$$KN = 0.066,\ TDAT = 14 \text{ msec},\ TACK = 6 \text{ msec}.$$

Case 1. VAX to SUN transfer

Apply equation (A7):

NP < = WS − 1.02 − 0.8426 − 0.966 with NW = 2.0 (approx.)
= WS − 2.829.

This implies that a cushion of 2.829 buffers is needed at the receiver to keep the SUN running continuously with a minimal flow of acknowledgement packets.

With ws = 15 buffers, α(SUN) = 0.81,
With ws = 8 buffers, α(SUN) = 0.65.

Case 2. SUN to VAX transfer

Apply Equation (A4)
NP < = WS − 0.966 − 0.824 − 0.414 with NW = 2.0
= WS − 2.204.

This implies that a cushion of 2.204 buffers is to be provided at the receiver to keep a continuous flow of packets.

With ws = 15 buffers, α(VAX) = 0.853,
With ws = 8 buffers, α(VAX) = 0.7245.

APPENDIX B: DESCRIPTION OF LNTP (LOCAL NETWORK TRANSPORT PROTOCOL)

A brief description of the new protocol is given below (the connection set-up and close-down sequences are not described here; please refer to [28] for details):

B.1 Protocol structure

The structure of the packets handled by the protocol is as shown in figures B.1 and B.2.

SRC PORT	DSTN PORT	PKT ID	LEN	SEQ NO	RETX CNT	DATA	CHK SUM

Fig. B.1. Structure of a DATA and DATA + PRMPT packet

SRC PORT	DSTN PORT	PKT ID	SEQ NO	CHK SUM

Fig. B.2. Structure of a control packet

Let us denote the components of a packet by a "C"-like structure

```
pkt = struct
      {
      SRCPORT     16 bits;
      DSTNPORT    16 bits;
      PKTID       4 bits;
      LEN         12 bits;
      SEQNO       4 bits;
      RETXCNT     4 bits;
      DATA        (LEN) bytes;
      CKSUM       16 bits;
      }
```

pkt.SRCPORT --> Port address associated with the sending peer
pkt.DSTNPORT --> Port address associated with the receiving peer
pkt.PKTID --> Packet identifier. Identifies DATA, DATA + PRMPT & PRMPT from sender; RETXMT, RESUME, HOLD from receiver
pkt.LEN --> Data length in octets
pkt.SEQNO --> Sequence number of the data packet from sender. Wrap around occurs after every MAX_SEQNO packets. From the receiver, this is one plus the sequence number of the packet last received correctly
pkt.RETXCNT --> Retransmission count of a data packet (zero initially)
pkt.DATA --> Data
pkt.CKSUM --> Checksum of the entire packet, including header

The variables describing the protocol machine are

MAX_SEQNO --> Maximum sequence number space (16)

Sender

SND.NXT --> Sequence number of the next packet to be sent
SND.UNA --> Sequence number of the first packet in the sender waiting to be acknowledged
Ntx --> Sender's threshhold in the protocol window

The following entities are also defined

1. A timer associated with the DATA + PRMPT & PRMPT packets in the flow control region.
2. A timer at the sender on every outgoing packet (TS) in the non-flow controlled region; it is reset on the arrival of either the next *outgoing*

data packet (it restarts the timer) or an acknowledgement packet. Similarly there is a timer (TR) at the receiver on every incoming packet; it is restarted by the arrival of the next *incoming* data packet. The expiry of TS generates a DATA + PRMPT from the sender. The expiry of TR generates an acknowledgement at the receiver. The timer values are chosen such that

$$TS > ETT(DAT) + TR + ETT(ACK),$$

where ETT is the end-to-end transfer time for a packet. TR has to be chosen appropriately. The rationale behind this timer structure is that for stream traffic, the next outgoing/incoming packet resets TS/TR so that no extra network traffic is generated. For interactive type traffic the timer values satisfying the above inequality will guard against possible deadlocks that might arise owing to packet losses.

3. A queue containing assembled packets waiting to be transmitted.

Receiver

RCV.NXT -->	Sequence number of the next packet to be received
RCV.CNS -->	Sequence number of the first packet in the receiver waiting to be consumed
NTR1 -->	Threshhold value when the window decreases
NTR2 -->	Threshhold value when the window increases (NTR2 < NTR1)

The sending peer can be in one of three states:

1. NORMAL state
2. PRMPT_SENT state
3. HOLD state.

In the NORMAL state only data packets flow to the receiving peer. The transition to PRMPT_SENT when the window size becomes less than NTX provokes a control packet. In this state the protocol can still accept packets from the client layer and send DATA + PRMPT packets. In the HOLD state no packet is sent except those provoked by RETXMT(n) control packet.

The type of packets from the sending peer are

1. DATA packets
2. DATA + PRMPT packet
3. PRMPT control packet.

The function of the PRMPT packet is to force an acknowledgment packet from the receiving peer.

The receiving peer can exist in one of three states:

1. NORMAL state
2. RETXMT_SENT state
3. HOLD_SENT state.

In the NORMAL state, packets are accepted without generating any acknowledgment unless an out-of-order packet arrives which provokes an immediate RETXMT(n) control packet. Packets are still accepted by the protocol in the HOLD_SENT state until the window closes.

Three types of packets may be sent by the receiving peer:

1. RETXMT(n) packet
2. RESUME(n) packet
3. HOLD (n) packet,

where $n - 1$ is the sequence number of the last packet that has been received correctly. Their functions are as follows:

RETXMT(n) -->	Retransmits packet of sequence number n. This is sent when an out-of-order packet arrives at this layer. The arrived packet is buffered.
RESUME(n) -->	Resumes transmission starting from packet of sequence number n. This is used when sufficient window space is available to re-enable the sending peer. Also this is sent in response to a PRMPT control packet when packets up to $n - 1$ have been received correctly and the receiver is expecting packet n.
HOLD(n) -->	Instructs the sending peer to hold transmission until instructed to resume. This is sent when the window size falls below a threshhold.

B1.1 Error recovery

The recovery from the various error conditions are performed as indicated:

Out-of-order packets -->	By means of RETXMT control packets used to initiate selective retransmissions and sequence numbers assigned to packets

Damaged packets --> By means of checksum validation and RETXMT packets

Duplicates --> By means of RETX_CNT assigned to packets

B1.2 Flow control

This is achieved by means of PRMPT packets from the sender, and HOLD and RESUME packets from the receiver.

The sender's window is defined as

SND.NXT <= sequence number < SND. UNA.

Sender's window size (SWS) = MAX_SEQNO − (SND.NXT − SND.UNA).

Two regions are defined:

SWS < NTX ... Region 1
SWS >= NTX ... Region 2

No flow control is initiated from the sender in Region 1. Once the sender enters Region 2, it initiates flow control measures maintaining the data flow at the same time until the protocol stops when the window is filled.

The receiver's window is defined as

RCV.NXT <= sequence number < RCV.CNS.

Receiver's window size (RWS) = MAX_SEQNO − (RCV.NXT − RCV.CNS).

Two regions are defined for RWS:

< NTR1 ... Region 1
>= NTR1 ... Region 2
when in the NORMAL state, and
< NTR2 ... Region 1
>= NTR2 ... Region 2
when in the HOLD_SENT state.
(In all cases, NTR2 < NTR1 <MAX_SEQNO.)

The point at which flow control takes effect

= [Time at which RWS crosses NTR1]
or
[Time at which SWS crosses NTX]

The hysteresis (NTR1 − NTR2) is needed to absorb any transient surge in packet arrivals when the sender moves from the HOLD state to the NORMAL state, thereby avoiding any ping-pong effect.

References

(1) S.R. Bunch and J.D. Day, "Control structure overhead in TCP," *IEEE Computer Networking Symposium*, May 1980, 121–7

(2) "DARPA Internet program protocol specifications," *Transmission control protocol*. RFC793, Information Sciences Institute, USC, CA, Sept. 1981

(3) "DARPA Internet program protocol specifications," *Internet protocol*. RFC791, Information Sciences Institute, USC, CA, Sept. 1981

(4) David D. Clarke, *Modularity and efficiency in protocol implementation*, RFC817, MIT Lab for Computer Science, July 1982

(5) C.J. Bennett and A.J. Hinchley, "Measurements of the Transmission control protocol," Computer Networks, 2 (1978), 396–408

(6) E. Arthur, G.L. Chesson, and B.W. Stuck, "Theoretical performance analysis of sliding window link level flow control for a local area network," *Proc. of the 8th Data Comm. Symposium*, Oct. 1983, 95–100

(7) L. Kleinrock, "Certain analytic results for time-shared processors," *Proc. IFIP Congress 1968*. Amsterdam: North-Holland, 838–45

(8) D.R. Cheriton "Local networking and internetworking in the V-System," *Proc. of the 8th Data Comm. Symposium*, Oct. 1983, 9–16

(9) Hideyuki Tokuda and Eric G. Manning, "An interprocess communication model for distributed software testbed," *Proc. of the ACM SIGCOMM '83 Symposium*, March 1983, 205–12

(10) 3COM Corporation, *Multibus Ethernet Controller reference manual*, May 1982.

(11) 3COM Corporation, *Unibus Ethernet Controller refernce manual*, 1982

(12) Digitial Equipment Corporation, *Networks and Communications Catalog*, Summer 1984

(13) Mail distributions from ARPANET TCP/IP mailing list, Sept.-Oct. 1984

(14) B.W. Lampson, M. Paul, and H.J. Siegart, *Distributed Systems: architecture and implementation*. Springer-Verlag, 1981

(15) A.Z. Spector, "Performing remote operations efficiently on a Local computer network," CACM, 25: 4 (April 1982), 246–60

(16) J.F. Shoch, and J.A. Hupp, "Measured performance of an Ethernet Local computer network," CACM, 23: 12 (Dec. 1980), 711–21

(17) M. Marathe, and B. Hawe, "Predicted capacity of Ethernet in a university environment," *Introduction to Local Area Networks*, DEC publications, 1982, 145–59

(18) E.E. Balkovich, and J.A. Morse, "Performance of distributed software implemented by a contention bus," *Proc. of Symp. on Data Comm., ACM SIGCOMM*, Nov. 1981, 39–45

(19) J.R. Spirn, J. Chien, and W. Hawe, "Bursty traffic local network modelling," IEEE Journal on Selected Areas in Communications, SAC-2: 1 (Jan. 1984), 250–7

(20) R.P.A. Collinson, "The Cambridge Ring and UNIX," Software practice and experience, 12: 12 (Dec. 1982), 583–94

(21) International Standards Organisation, *Open Systems Interconnection*, ISO/TC97/SC16, June 1982

(22) D. Ferrari, *Computer Systems Performance Evaluation*. Englewood Cliffs, NJ: Prentice-Hall, 1978

(23) J.F. Schoch, et al., "Evolution of the Ethernet local computer network," IEEE Computer, 15: 8 (Aug. 1982), 10–27

(24) R.M. Metcalfe, and D.R. Boggs, "Ethernet: distributed packet switching for local computer networks," CACM, 19: 7 (July 1976) 395–404

(25) D.R. Boggs, et al., "PUP: an internetwork architecture," IEEE Trans. on Communications, COM-28: 4 (April 1980), 612–23
(26) R.M. Needham, and A.J. Herbert, *The Cambridge Distributed Computing System*. Addison-Wesley Pub. Co., 1982
(27) I.W. Cotton, "Technologies for Local area computer networks," Computer Networks, North-Holland Pub. Co., 4 (1980), 197–208
(28) S. Chanson, K. Ravindran, and S. Atkins, "LNTP – An efficient transport protocol for Local Area Networks," Technical Report TR85-4, Dept. of Computer Science, University of British Columbia, Dec. 1984

SAMUEL T. CHANSON is an associate professor in the Department of Computer Science at the University of British Columbia, Vancouver, BC, Canada. Prior to joining U BC he was an assistant professor in the School of Electrical Engineering at Purdue University, W. Lafayette, Indiana. He holds the B SC degree in Electrical Engineering from Hong Kong University, the M SC and PH D degrees in Computer Science from the University of California at Berkeley. He has published in IEEE transactions and many other journals and conference proceedings. His research interests include computer communications, especially local area networks, performance evaluation of centralized and distributed computing systems, and operating system designs.

K. RAVINDRAN received his B ENG degree in Electronics and Communications Engineering in 1976 and M ENG degree in Automation in 1978, both from the Indian Institute of Science, Bangalore. From 1978 to 1982 he was a Systems Engineer at the ISRO Satellite Centre, Bangalore. Currently, he is working towards his PH D degree in the Dept. of Computer Science at the University of British Columbia, Vancouver, BC, Canada. His research interests include the design and performance evaluation of distributed systems, computer networks, and real-time systems.

STELLA ATKINS recently joined the Faculty of Computer Science at Simon Fraser University, British Columbia, teaching computer networks and operating systems. She has done postgraduate studies at Warwick University, England, and is completing her PH D at the University of British Columbia on the subject of exception handling in distributed operating systems. In Canada Ms Atkins has worked with a computer consultant, specializing in technical computing for the forest industry and in computer simulation models. Her research interests are in the areas of multiple-process operating systems, computer communications, and local area networks.