The Influence of Protocol Choice on Network Performance

By Brendon J. Whateley

Table of Contents

List of Figures

List of Tables

Chapter 1 Introduction

Computer communication networks are a vital link in providing many of the services that we use daily, and our reliance on these networks is on the increase. The growing use of networks is driving network design towards greater performance. The greater need for network connectivity and increased performance makes the study of network performance constraints important [19].

Networks consist of both hardware and software components. Currently great advances are being made in network hardware, resulting in advances in the available raw network performance. In this thesis, I will show through measurement that it is difficult to harness all the raw performance and to make it available to carry network services. I will also identify some of the factors limiting the full utilization of a high speed network.

Background

Protocols serve to hide the physical characteristics of a network from applications and provide an abstraction that makes the network more useful to these applications. Network protocols provide the required level of service to the applications that use the network. The protocols perform a number of tasks, such as: taking care of the reliable, correct delivery of data, and optimizing the use of system resources.

Protocols are required to ensure that communication between machines and remote locations is successful, despite the uncontrollable environment through which communication takes place [25]. Networks consist of a large infrastructure and many components that are

distributed outside of the computers that they link together. The external location of many network components places them outside the controlled computer's environment, which exposes these components to uncontrolled interference. The nature of large networks usually requires that parts of the network are supplied by external organizations, which further removes control of the environment and quality of the service.

Protocols provide the means of ensuring that correct communications take place even when the external environment causes errors in the data traveling in the network. Several types of damage can be incurred by data as it travels through the network. Data can be missing, data can be present but altered, or data can be received in the wrong order [25]. More than one type of damage can occur to the same data at the same time. The protocols have to detect when received data is different from the sent data and take steps to correct the damaged data.

The environment may also contain other hazards. These include deliberate attempts to disrupt the network or attempts to gain unauthorized access to the network and the services that are available on the network. If these threats are present, it is the responsibility of the protocol to make such attempts as difficult as possible.

Networks suffer from other physical constraints. Available network components and technology may not be ideal to support the application's communication requirements. Protocols have to make the best use of the available physical resources. These resources include memory system bandwidth, CPU performance, network component performance and interface performance. Performance problems generally

occur as data moves across interfaces between different types of hardware, such as moving from the computer's memory to the network interface. The computer architecture can limit a machine's ability to move data from its memory to the network interface.

Further resource constraints are introduced whenever public networks are used, removing parts of the overall network from the control of the network designer. This may provide various constraints on the overall network that conflict with applications requirements and constraints.

Existing protocols and continued protocol research are able to address all the above issues to some degree. The problem is that various protocols address different aspects of the problem, but often do not address all the relevant concerns of an application. Even fewer protocols address the newer requirements to produce high performance from existing interface technology. The conflicting constraints that exist in network protocol design results in applications having to compromise on which issues they require the protocol to handle.

Many different network protocols exist due to the difficulty involved in designing a single protocol to meet the wide range of application requirements while being constrained by the underlying network limitations. Often the application requirements conflict with each other. For example, it may be necessary to make a trade-off between protocol functionality and performance. Simple solutions to these issues do not currently exist and given the physical systems that are used to build networks, it is unlikely that the situation will improve in the near future.

Existing equipment has to be included in most networks. This equipment does not always provide the hardware support needed for efficiently meeting the needs of the protocols. This poses additional constraints on the performance of protocols. As an example, people still wish to use 1200 baud modems that they own. This is despite the existence of much higher speed equipment. In the same way, high speed network equipment has to function in conjunction with hardware which is not designed to meet such a network's performance requirements.

Political and business considerations create situations in which existing hardware has to be used or less than ideal technology gets chosen for political reasons. These factors can be considered to be additional constraints under which the protocol has to perform. These are reasons why network and protocol designs cannot only concentrate on the state of the art systems. Real systems have to be built to coexist with existing technology.

Outline

My aim in this thesis is to investigate the problems preventing the full use of modern high speed network systems when using existing equipment. To do this, I measure the performance of several protocols that make use of high speed network hardware. I will use this data to isolate some problem areas with current software and hardware designs. I will show that getting higher performance from networks is not simply a matter of building faster networks. I will also show that there exist performance mismatches in current network interface systems. Using this information, I will suggest some possible solutions to these performance mismatches. My measurements are supported by

practical design and coding experience on the same high speed network equipment.

This work is divided into several sections:

- Chapter 2 explains the background and reason for this thesis and discusses some relevant related work.
- Chapter 3 outlines the reasons for studying protocols and discusses several different protocols that were examined.
- Chapter 4 has the performance measurements and discusses the meaning of the various results and the artifacts inherent in the data. This chapter concludes with a description of the implementation work I undertook to support the measurements.
- Chapter 5 discusses how the implications gained from the measurements and implementation experience could be used to plan solutions to the problems observed. Chapter 5 ends with a summary of the results and conclusion.

Chapter 2 Background and Related Work

Computer communication takes place when a message is successfully transported from a sending application to one or more receiving applications, possibly on other machines. While this is a simple idea, many hardware and software components are involved and many transient problems caused by the environment can hamper this communication. In this thesis, I will concentrate on how the software that implements the protocol interacts with the computer and network hardware. This software controls the movement of data from the memory of the sending application to the network, and from the network into the memory of the receiving application.

The physical components of the network consist of hardware to convert the data into a format compatible with transmission over the network medium [25]. This process usually involves the modulation of some carrier signal. The modulated carrier is used to carry the data over a wire (or other medium) to the destination. The data is extracted from the modulated signal and made available to the protocol software on the receiving side. Higher speed networking hardware increases the rate at which data can be transformed and transmitted over the network medium. I will show that current hardware and software do not always allow the hardware performance to be realized by applications.

As discussed in the introduction, protocols can be designed to offer a variety of features to applications. This has resulted in a number of different protocols being developed. In addition to the number of fea-

tures that can be implemented, each can be implemented in several ways. Each implementation has to face several trade-offs. The trade-offs are the result of the often conflicting requirements of the various possible features. An example would be a feature such as reliability which increases implementation complexity and reduces performance through the use of acknowledgment messages and other control information.

In this thesis, I am primarily concerned with the end-to-end application level performance of the network system and possible reasons for not achieving maximum performance. The performance of a communication system can be measured in terms of network latency and communication bandwidth.

The bandwidth is the amount of information that can be transported by the network in a given amount of time [25]. The bandwidth determines the amount of time it takes to send a given amount of data.

The propagation delay is determined by the physical distance between the source and destination, and by the maximum signal propagation speed, which is the speed of light. In practice, the propagation speed is less than the speed of light due to different transmission properties of the materials used to make up networks. This propagation delay is approximately 1 nanosecond per foot of transmission distance in copper wire. If satellite links are part of the network, the long distances involved introduce long propagation delays.

Latency is the length of time it takes for messages to travel from source to destination. Protocol acknowledgment messages are subject to similar delays. Latency is the result of propagation delay of the data

signal, data transmission time, queuing delays, and protocol processing time.

Queuing delay is the time that messages spend on queues at various nodes in the network. This results from conflicts regarding network resources, messages are queued pending availability of a network link or some other resource.

Processing time is the length of time required to process messages while traversing the network. It includes computation of checksums, error correcting and interpreting of header information.

The transmission rate of the network is its ability to put data on the physical medium and is measured in bits per second. The transmission rate determines how fast messages can be sent.

The transmission time is the time taken to send a message. It depends on the underlying transmission rate and the size of the message. Dividing the message size in bit by the transmission rate in bits per second yields the transmission time for the message.

The data rate is the number of bytes of user data that can be sent every second. This is related to transmission rate, but is very different. Various network layers add headers, trailers, checksum information and error correcting codes to the user data before it is placed on the network. The data rate is a more useful measure network performance from an applications perspective. Throughput is another measure of how many bytes of user data can be sent each second.

The protocol influences the data rate of the network as well as having an influence on the latency. It does this by reducing the data rate through the addition of data in the form of headers, trailers and error

detection and correction information, as well as through the transmission of protocol messages. The act of sending this extra data reduces the amount of bandwidth devoted to carrying the application data. Latency is increased through a combination of the extra transmission time, processing time and time spent waiting for protocol messages. Different protocols effect overall performance to different extents and in different ways.

Different network designs produce different characteristics. Satellite links are an example of a network link that impose large propagation delays due to the long distances involved. Network characteristics such as this can favor some protocols more than others. There are also interactions between basic network characteristics. For example a network with very short propagation delays but with a low transmission rate may be dominated by the long transmission times which may exceed the propagation delay orders of magnitude. In this case longer propagation delays would have little effect on the overall performance.

A large bandwidth can result in a reduction in latency if all other factors remain constant. This is achieved in two ways. First the transmission time for a message will be reduced, which reduces the time for the receipt of an entire message after transmission has begun, particularly if the message is received and resent many times at intermediate nodes in the networks. The second is the reduction in congestion on any heavily used network link. Congestion causes messages to be placed on queues at various stages throughout the network, which adds to the time it takes for the message to reach its destination. In addition to increasing the delay by increased message buffering, messages can be lost by the network when congestion is experienced. This

loss is due to the limited resources in the network, which at some point result in network packets being dropped. This is a behavior of almost all wide area networks. Other network designs have been proposed that eliminate the need to drop packets [15] but few are in widespread use and all have other undesirable properties. A typical undesirable property is to spread the congestion and slow down packets that were never destined to go through the congested portion of the network.

Each application that uses a network has a wall-clock time that is the maximum acceptable time for the completion of a particular transaction. If a transaction takes longer to complete than this acceptable time, the communication system is too slow. The performance criteria vary according to the particular application. Some applications require very low latency: an example would be a simple terminal application that requires characters to be echoed as they are typed. For such an application, a one second latency would be unacceptable. The application's data rate requirement would be low, 10 characters per second would be more than enough for typing (computer output would however benefit from rates of 2000 characters per second or more when updating a screen.) Other applications require high bandwidth with little regard for latency. Examples of this would be most applications that transfer large amounts of data such as file transfer programs connecting to FTP servers. As application communication requirements have become more complex, there has been an increasing demand for high speed networks. That is, networks with higher bandwidth and lower latency.

Several other factors are also important in the discussion of protocols. Most of these can be considered as issues related to the quality of ser-

vice offered by the communication system. These are value added services that can be added in a variety of ways and at various levels in the protocol stack, including in the applications. Reliable or guaranteed delivery can be built on top of an unreliable basic protocol such as UDP. For reasons of ease of use and performance, building these features into the correct layers of a protocol stack is important. This is the same reason why C programs use C libraries instead of reinventing every function from some set of primitives. It makes sense to invest the effort required to build the desired functionality only once, and then reuse it every time it is needed. This allows the elimination of the duplication of work that would otherwise be required. It also means that bugs only have to be located and corrected once, rather than have the potential for causing the same problems in every application.

Related Work

Soon after computers became useful to organizations, it became apparent that information needed to be moved from one machine to another. This requirement started the development of computer networks. Computer networks consist of two components, a hardware or physical component which carries the communication signals from machine to machine, and a software component which controls the use of the physical medium, the transmission of data, and the level of service that the network is able to provide. These two areas are researched separately, yet are very closely coupled in a functioning system. The interaction between the two systems is key to realizing the full potential of the network system as a whole.

Traditionally the controlling protocols have been designed with a stack architecture. The stack consists of a series of layers piled on top of each other. Each layer communicates only with the layer above and below, passing data through after performing some transformation or function on or with the data. A standard model is the International Standards Organization's (the ISO's) Open Systems Interconnect (OSI) model.

The ISO's OSI model consists of 7 distinct layers, each serving a unique role [25]. The layers are determined by applying the following methods to the range of tasks that need to be accomplished by the protocol software:

- A layer is needed for each different level of abstraction that can be found in the system.
- Each layer should perform a well defined function.
- The layers should be chosen to create a set of standard functional layers across distinct protocols. In other words, the layers of the protocol stack should be made independent.
- The layers should be designed in such a way as to minimize the flow of control and or context information between the layers. This implies that the layers should be designed with the simplest possible inter-layer interface possible.
- The number of layers should result in enough layers to contain only one major function per layer but the number of layers should not be so large as to be impractical.

The layers proposed by the ISO are as follows:

The Physical Layer

This layer is involved with the transmission of a raw data stream. It involves the physical medium on which the network is constructed, such as the wire, optical fiber, radio and satellite links. Its function is to carry stream of data bits from place to place.

The Data Link Layer

This layer involves the transmission of data frames. Its function is to remove errors that may be introduced in the physical layer. It does this by requesting retransmission of data frames and by rejecting duplicate frames. It also has the ability to implement some flow control.

The Network Layer

This layer deals with routing packets to the correct destination. The destination can be on the same machine, in which case a message is passed up the stack to the transport level, or it may be for a remote machine, in which case the message is passed to the data link level for a new destination machine.

The Transport Layer

This layer manages network connections, acting as the source or destination for data traveling through the network. It handles message disassembly and reassembly, handling size mismatches between higher levels and lower levels in the protocol stack. The transport layer provides end-to-end reliable connectivity. It provides the flow control and the error control functionality which ensures the reliable delivery of data.

The Session Layer

This layer negotiates network connections, establishing a context or session in which communication takes place.

The Presentation Layer

This layer performs data transformation. It takes care of byte ordering issues between host architecture and network architecture, as well as possible data encryption or compression.

The Application Layer

This layer deals with user application design issues.

This functionality is required in all network systems. (In some systems some of the functionality may be available without any action being taken.) As the trend in network design has moved towards faster networks, the challenge has become the need to perform these tasks fast enough so as not to waste physical bandwidth at the physical level of the network. As networks increase in performance, it becomes increasingly more difficult to implement protocol stacks in a layered approach. The reason is simply that layered implementations perform poorly [19].

Early Networks

The early wide area networks were started in the 1960's. ARPANET simulation started in 1960 with actual network implementation by 1969 [25]. This supported TCP as well as NCP protocols. SNA was

originally released in 1974, followed by later versions in 1976 and 1979. The subsequent versions supported greater interconnectivity.

As networks have grown, performance has increased. Increasing use has been made of X.25 (As an example, I have recently designed and implemented an asynchronous transport protocol for a commercial application that makes use of dial-up X.25 data lines) and packet frame relay. The benefits shown by the older network technology has fueled great interest in newer high performance technology such as ATM protocols. There is growing commercial acceptance of the need for wide area networks. This acceptance is related to the increasing need to have access to data that may be distributed around the world.

In Search of High Speeds

Studies of processor performance as it relates to networking show that copying data is getting proportionally slower as processors get faster [17]. This behavior is the result of memory speeds not keeping up with high end processor clock speeds. The very high cost of memory that can match CPU clock speeds has resulted in a move towards caching architectures. These make use of small amounts of high speed memory close to the processor with layers of slower and larger memory further away from the processor.

Caches work by exploiting locality of data and instruction references by keeping frequently referenced memory addresses in a small fast memory. The memory system tries to fulfil memory references from this fast memory before accessing the large slower main memory. Unfortunately network activity, by its asynchronous nature, causes code usage to violate the locality assumption. This causes the instruction cache to need reloading every time a new packet arrives. The

nature of the data copying required also does not present an opportunity to make good use of a cache. The amount of data to be accessed is large, and the number of accesses to each location is small. This results in the CPU being slowed to the speed of main memory, which can result in a slowdown by a factor of 4 or more [19].

A lot of effort goes into designing operating system and network software that minimizes the need to copy data. One such approach tries to use a small amount of hardware support to gain a lot in performance. I was involved in the design and implementation of a memory based messaging system.

Memory Based Messaging

Memory based messaging is a technique that uses the memory and hardware of the machines involved in the network in a new and unique way. It can be used as the kernel to application interface in micro-kernel systems, removing the network code functionality from the kernel. This can be done because of the simple support which is required of the kernel to support memory based messaging.

Memory based messaging is implemented using network hardware which is designed to take data out of application memory on transmission, and deliver it into application memory on reception. The transmission of data is triggered by some application activity such as writing to memory or signalling the complete setup of a block of memory. In effect, this model gives the appearance of shared memory between processes. This shared memory effect may be one-way, in which case one machine may write to the memory and the other machines can read the results but not update the original machine's memory, or it can be symmetrical, in which case each machine can update or read

the same shared memory area. This sharing of memory does not have to be one-to-one, but may be set up to be one-to-many, many-to-many or many-to-one. Each of these arrangements offers its own set of challenges [4].

These implementation and protocol challenges are mostly based on the fact that the memory only appears to be shared, thus producing a variety of concurrence issues. Each of these issues can be dealt with at several levels from in the hardware to in the software implementation of the protocol, or even application design. This is a new way of handling communication that offers many exciting possibilities.

Protocols needed to handle general purpose memory based messaging implementations can involve many protocol issues that are applicable to more conventional networks. Memory based messaging does demonstrate the protocol overhead that is imposed by the need to do the various housekeeping tasks that are required at the transport level. Tasks such as packet reordering and error correction can be very expensive, especially when no hardware is available to help.

Chapter 3 Study of Protocols

In this thesis I study two protocols (VMTP and ATM), to understand why it is not always possible to attain the high performance that would be expected from the underlying hardware. The problem I wish to address is that the existence of high speed hardware does not directly translate into high speed network performance at the application level.

To investigate this problem, the two most important network performance characteristics were evaluated for the two protocols. These performance characteristics are throughput and latency. The first is a measure of the amount of data that can be sent through a network in a given time. The second is a measure of how long it takes for a particular message to cross the network.

These two measures of performance are important because they address the two most important questions that an application programer may ask of the network, 'how fast can I send data?' and 'how long will it take to get a reply from a remote machine?'. New applications such as real time video and HDTV place increasing demands on network performance. It must also be remembered that networks are seldom used by only one application, so the performance must be satisfactory while meeting the various needs of multiple applications.

Protocol Evaluation Criteria

The ultimate test of a protocol is whether its performance meets the requirements of the applications that wish to use it. This is determined by the interaction of the protocol with the network and computer

hardware, given the particular workloads experienced by the components of the system. To make the comparison of protocols generally applicable, I decided to perform the measurements at a protocol level instead of at an application level. The speed of data movement is measured in both the amount of data that can be moved in a given time (throughput) and the length of time it takes for a message to get to its destination (latency).

There are many protocol requirements that result in overhead that reduces the available bandwidth. Protocols have to reach a compromise which allows the overhead to be kept to a minimum while allowing all the required protocol functionality to be retained. Packet headers, trailers and checksums are usually required to handle the packet address, protocol specification and error checking. This information has to be added to each transmitted packet. A small packet size, such as used by ATM, results in more packets, leading to increased protocol overhead.

The need to buffer packets in routers and switches adds to the time it takes for packets to get through the network. Techniques such as cut-through routing which provided significant gains in latency at lower network speeds, provide significantly less benefit at higher network speeds. Cut-through routing provides a benefit by partially reducing the buffering time and cost while a network node is forced to wait for the reception of the complete message from a link before forwarding it. As link transmission speed increases, the time taken to receive an entire message decreases, reducing the potential benefit. Congestion prevents cut-through routing from providing any benefit since congestion forces packets to be buffered and sometimes dropped. Dropped packets require parts of a message to be retransmitted, which reduces

throughput by using more bandwidth and also increases message latency. How protocols deal with these issues is of great importance when considering various protocol designs.

Network latency depends on several factors. The time taken to process the data before sending it; the actual transmission time on the transmission medium; the propagation delay, the time to buffer, process and forward the packets at each router or gateway; and the time to process the packets at the receiving end to extract the data. This time is doubled if an acknowledge has to be sent back in reply. Latency is also influenced by hardware constraints such as bandwidth of links, buffer space in gateways and routers. The method of routing that is used can also influence the length of the delay in gateways and routers.

The reliability of communication depends on the efforts the transport protocol and lower layers put into ensuring timely and accurate delivery of the packets in the data stream. Mechanisms can include retransmission system, error detection and/or correction methods using the transmission of redundant data. Each of these methods involves tradeoff between performance and resource utilization.

Security can have great implications in terms of performance. If some form of encryption is used, the time taken to encrypt and decrypt the data can be a significant overhead on the cost of the communication. The impact of packet loss on the applications can also be affected by the method of encryption used. For example a chained encoding system would be unable to decode any data received after a packet loss unless some other method of resynchronizing is employed, again at some additional cost [8].

Protocol complexity can make implementation difficult, software reliability difficult or even impossible to ensure, and it can also add to the time taken to process data before sending or delivering the data. Some level of complexity is required to provide the services that are required, but there is the potential of having to live with costs of features that are not required.

Overview of Protocols Studied

The two protocols that were studied form different layers of the OSI protocol stack, and as such perform different functions. The similarities and differences between the protocols results from the particular problems each attempts to address. The usefulness of each protocol is determined by its ability to deliver the correct functionality to the applications while making efficient use of the underlying network. The following sections give a more detailed introduction to each protocol.

The VMTP Transport Protocol

VMTP is an end to end request-response protocol that was designed for a distributed system such as the V operating system [11, 7]. It forms the OSI transport layer in the protocol stack. VMTP is designed to support a client-server communication model. The V operating system is designed around such a client-server model and VMTP is therefore very well suited to the V operating system.

A VMTP transaction begins with a client sending a request to a server, possibly using multicast (one to many addressing). The server acts upon the request and then sends a reply back to the client. No separate acknowledgment is sent to the client unless one is requested by the cli-

ent. Reliability is provided by a time-out driven retransmission mechanism. This mechanism works well in situations where packet loss is not common, but does result in significant cost when failure without response is a common network behavior. This type of good behavior for the protocol in some situations and very poor performance in others illustrates the design trade-off that is involved.

The VMTP implementation becomes complex when trying to support some of the more advanced protocol functionality. One such advanced function is VMTP's streaming mode. In streaming mode, the amount of data that can be sent in a single message is greatly increased from 16 kilobytes to 4 megabytes. This adds considerable complexity to the buffer management and error recovery that are needed to handle various exception conditions that may arise. A large part of this problem would apply to most protocol implementations. The protocol has to keep all data sent until it is acknowledged by the recipient (or recipients in the case of multicast). This is required to allow for the possible retransmission of packets that may be lost in the network. This results in every stream that is being transmitted or received needing to reserve up to 4 megabytes of kernel memory, which for performance reasons has to be physical memory instead of virtual memory. This quickly exceeds the memory that is available within most systems kernels.

Multicast or broadcast is another advanced feature which often adds a performance and a resource overhead to every participant on the network. This arises out of the practical need for each machine to receive a multicast message and allow for the possible delivery to processes on that machine. Although some network interfaces can support multicast directly (e.g. ethernet,) it may be impossible to avoid copying

the message into main memory before deciding if a message is destined for an application on a particular machine. This process costs the wrongful recipient in terms of wasted memory system bandwidth and CPU cycles that could have been used for application processing. This, as we shall see later, can become a severe limitation as network performance increase. It is difficult to eliminate this problem without incurring other costs. A possible solution would be the use of network adaptors that could resolve the addressing issues before the message is copied into main memory, but it is not necessarily a desirable solution to require all machines to have special hardware.

It seems that for every network protocol, the difficulties and problems arise not in the normal operation but in the gray areas where errors and odd conditions have to be handled. If retransmission of data was not required, then no buffers would be needed in the kernel to keep the data around. A much smaller set of buffers would be required to formulate the required packets for transmission over the network. The reality of uncontrollable events results in the need for error detection and error correction facilities.

ATM

The Asynchronous Transfer Mode (ATM) protocol forms the network layer of the OSI protocol stack. An important feature of the ATM protocol is its small, fixed cell size. The fixed size cells have a number of desirable implication to both software and hardware design. The most notable is the ease of designing routers because of predictable resource requirements both in terms of transmission time and buffer requirements.

The small cell size means that almost all useful information must be sent in multiple cells. This leads to additional cost of the segmentation and reassembly mechanism that is required to split data into cells and the more expensive reassembly of cells into contiguous data. Small cell size also results in a larger number of headers for a given amount of data. In other words, when using ATM there is one cell header for every 48 bytes of data, cell headers are 5 bytes long (total cell size 53 bytes). This results in 9.5% of the available network capacity being used to transport cell headers instead of useful payload data.

The design of ATM seems to indicate that it will most often be used to carry some higher level protocol. This potentially leads to the duplication of protocol features such as error detection. Protocols used over ATM networks need to be aware of the services that the underlying network provides to avoid the cost of duplication.

Overview of ATM Cards

FORE Systems TCA-100 ATM cards using a TURBOchannel interface to the host DECstation were used. The cards provide hardware support for the ATM Adaption Layer (AAL) by using a Class 3/4 Segmentation And Reassembly (SAR) cell format. The SAR format could be manually overridden if required. If used, the SAR format consumed 2 bytes of each cell for a header containing sequence and message identification information and 2 bytes for payload length and CRC information.

The fundamental limitation of the ATM cards that I evaluated is that the kernel is expected to move the data to and from the card using software control. The reading and writing of data over the bus is expensive in terms of memory bandwidth. When reading or writing to

the interface card, the CPU cannot get any benefit from cache systems because the requirement is for the data to be placed in the interface card's registers or read from the interface card's registers.

In addition, the bandwidth of the bus on which the interface card is placed can be slower than memory speeds in some machines. Many busses have a burst transfer mode, and this includes memory systems that have a burst mode to fill cache lines. When using a card such as the FORE systems TCA-100 ATM cards, it is not possible to take any advantage of this type of bus speedup technique. Each word has the overhead of going through bus arbitration to acquire the bus, which wastes cycles on every single read or write.

It is not completely clear what a better method of access for an ATM interface would be. The problem is caused largely by the small size of a cell. This means that no large transfers can be used unless the interface card has the logic to split the packets up and formulate headers. The receiving end would require the logic to reassemble the packets into larger units.

The problem of having this logic on the interface card is that it limits the future protocol changes that can be made. It also requires that the transport protocol software has to repeat much of the work that is being done on the card, but at a higher level. The use of scatter-gather Direct Memory Access (DMA) would certainly help with this.

A scatter-gather DMA controller is a direct memory access controller that can read from several disjoint memory areas and/or write to several disjoint memory areas in a single operation. This allows cell headers and cell bodies to be stored in different memory areas, yet still copied in a single DMA transfer. In particular, a series of headers could be

created for a large transfer, with each being appended onto a single cells worth of data during transmission. This would eliminate the need to copy the data while assembling the cells before DMA transfer to the network interface card, resulting in a reduction in the amount of data movement required and also the amount of memory system access negotiation that is required. The ability to do scatter-gather DMA is built into the DECstation's TURBOchannel system architecture.

Another solution is to provide dedicated hardware to handle specific applications such as video. Video cells could be handled by hardware and placed in the display frame buffer without needing access to the CPU or the system memory bus.

The receiving side presents greater problems in any network where packet ordering is not preserved across the network. This means that packets can arrive for different applications and out of order at the receiver. This task is made easier since ATM virtual circuits preserve cell ordering.

The protocol has to identify the applications that require the packets and reassemble the various packets for the various applications. This is complicated by the fact that the receiver often has to keep timers on the packets that it is assembling to request retransmission of lost packets. When a part of a new message arrives, the ATM header has to be examined to determine the destination protocol or application, the size of the total message has to be determined so that the necessary memory can be secured, and the packet fragment has to be copied from the ATM cell and put in the correct part of the packet buffer. A

timer must then be set on the buffer and a note made of which parts have been received.

When a cell is received, the segmentation and reassembly code must be determined whether it contains data for an existing partial message, whether it is a duplicate, or whether it is part of a new message. If it contains data for an existing message, the timer has to be reset, and the new data copied into the correct place in the buffer. A note has to be made of which parts of the total message still have to be received. If the entire packet has been received, the buffer is handed up to the destination protocol or application.

If a damaged cell is received, the cell can be discarded and a request for retransmission can be sent to the sending computer, or an attempt can be made to correct single bit errors if a standard ATM class 3/4 SAR cell format is used. The latter is possible because a class 3/4 SAR cell has a separate data Cyclic Redundancy Check (CRC) and header CRC. This means that a cell can have a valid header but damaged data, allowing a negative acknowledgment message to be generated. The possibility of correcting single bit errors in the header would be useful if it was known that the payload portion of the cell was undamaged.

These issues need to be addressed whenever we deal with a high speed network implementation. Later we will look at some solutions to the memory bandwidth limitations of an ATM network system.

Chapter 4 Measurements

This section on measurements will begin with a discussion of the work done using ATM cells over a 155 megabit/second ATM network followed by a discussion of the measurements taken on the same machines using VMTP over a 10 megabit/second ethernet.

Equipment

Several machines were used during the development and testing of the various implementation aspects of my work, including VAX 4 processor firefly, DECstation 3100's, 2 different models of the DECstation 5000's, an SGI Indigo and an SGI 8 processor machine. The 2 models of the DECstation 5000 varied in processor speed: the DECstation 5000/240 has a 33MHz processor and the DECstation 5000/200 has a 25MHz processor. Most of the following measurements were taken from a pair of machines.

Measurement Machines

The machines used for these tests were a pair of DECstation 5000 machines. One machine was a DECstation 5000 model 240 while the other was a DECstation 5000 model 200. Both machines were running Ultrix V4.2a for the actual performance tests. The DECstation 5000/200 could also support the V and V++ operating systems. Some performance aspects of the system were measured on the V and V++ systems to gain an understanding of the performance impact the different operating system implementations made.

The difference between a DECstation 5000/200 and DECstation 5000/240 is primarily the processor speed. In the DECstation 5000/240 the

processor is mounted on a daughter board and runs at 33MHz while the DECstation 5000/200 has the processor mounted directly on the 25MHz motherboard. Both machines have 25MHz motherboards and are designed around a Turbo Channel architecture.

The machines were configured as follows. The DECstation 5000/200 machine had 256 megabytes of main memory installed, which allowed disk access to be avoided in some of the performance tests. The DECstation 5000/240 was fitted with 32 megabytes of main memory. Each machine was supplied with a local disk.

The tests that were dependent on network load were performed at night when the background load on the network was minimal. This load was monitored regularly during network performance tests to ensure that the background load remained insignificant. Results that were obtained when network load was a factor were discarded and the affected tests repeated. Each test was performed multiple times, the results were examined for possible strange values. Values were considered strange if they were not reproducible. Strange values were removed and the remaining values were then averaged. This approach was taken to minimize extraneous outside influence and should yield similar results to a statistical system which eliminates outlying datapoints and averages the remaining data.

Turbo Channel Architecture

The TURBOchannel is a synchronous bus with a 32 bit address/data path. The DECstation 5000/200 is capable of 90 megabyte/second DMA transfers, 87 megabyte/second during software controlled DMA. Uncached CPU reads from memory have a maximum bandwidth of 10 megabyte/second while CPU cache fill reads have a 30

megabyte/second bandwidth. Page writes have a 100 megabyte/second bandwidth, non-page writes a 20 megabyte/second bandwidth and partial writes a 9.1 megabyte/second bandwidth.

DMA 1-word reads are limited to 9.1 megabyte/second with 128 word reads capable of 92.8 megabyte/second. DMA 1-word writes have a bandwidth of 14.3 megabyte/second with 128 word writes having a bandwidth of 95.5 megabyte/second. The maximum size of a single DMA transfer is 128 bytes. Any operations (including accesses to the TCA-100[1] that are stalled due to full queues) that exceed the maximum bus time are aborted with a bus error.

The TURBOchannel offers scatter-gather DMA operations. These operations could be used to split cell header information from cell payload when packets are received and do the reverse when the cells are transmitted. This behavior, if used, would increase the ATM interface bandwidth by better use of the TURBOchannel bus.

The current TCA-100 interface offers no DMA support. Thus the possible performance increases by off-loading the data movement from the CPU to the DMA system are only speculation. It would seem that substantial gains could be had from this approach.

ATM Measurement

The design of the hardware and the speeds involved in the ATM interface causes the measurements to be specific to this particular interface. The following section will provide details of the ATM interface and discuss the implications of the design on the software. This will be fol-

1. The TCA-100 is the ATM interface card used in the ATM network performance measurement.

lowed by a detailed discussion of the software needed to drive the interface. The final section will give the results of the measurements made using this system and software.

The TCA-100 TURBOchannel ATM Interface

The TCA-100 system used for this study was the first version produced by FORE Systems, Inc. The card is supplied with software which supports network communication using the Internet protocol through a standard Berkeley socket interface under Ultrix. This driver software was discarded for the purposes of this study for two reasons. The IP mapping to ATM cells provided an unwanted overhead and restricted the amount of control over the interface. The second reason was speed: the given driver performed more slowly than initial attempts with an optimized driver.

It should also be pointed out that the performance tests resulted in the detection of some hardware problems within the interface cards, which then required replacement. These problems caused the machine to crash due to TURBOchannel bus access violations, which resulted in bus time-outs. The network card seemed to fail to release control of the bus under certain high stress performance tests. The replacement cards arrived after all the performance testing had been completed. A few informal tests did not seem to indicate any performance difference using the new cards, but it was not possible to repeat all the performance tests.

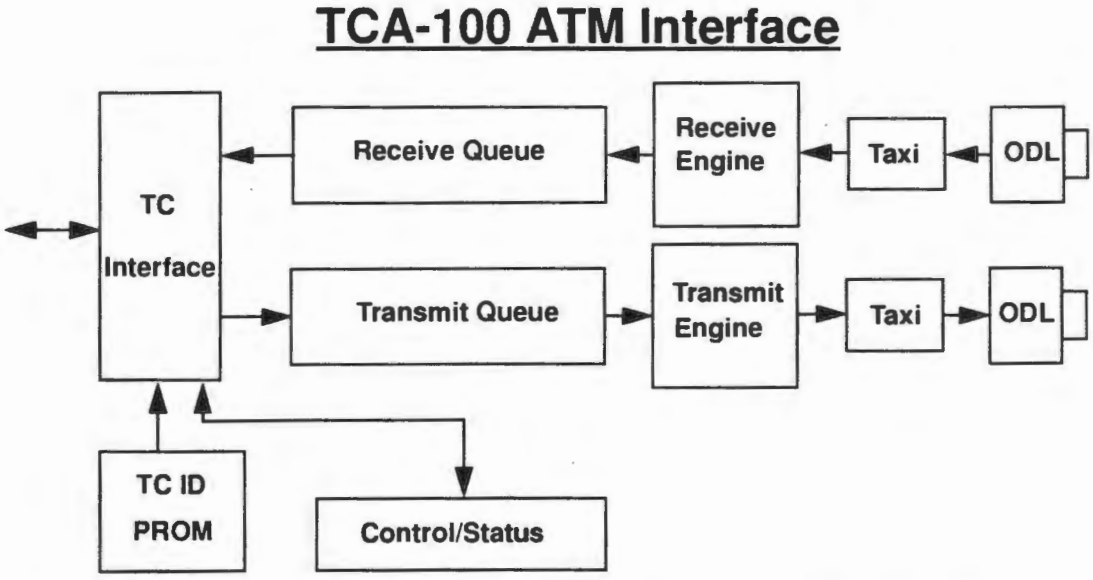
The card's interface is mapped into a 256 kilobyte region of the machine's address space. The driver that I wrote accessed this memory mapped interface through a non-cached portion of the processors address space. The design of the card makes the use of any direct

memory access (DMA) method of accessing the interface impossible. Therefore all data movement between the main memory and the card, both for data transmission and reception, has to be performed by the processor. These accesses also cannot make use of the TURBOchannel burst mode transfers. Each read or write, therefore, incurs the cost of TURBOchannel access arbitration. This significant overhead associated with each bus access increases the cost of each word transferred by several clock cycles.

TCA-100 Hardware Design

The card hooks up to two optical fibers, one for transmission and one for reception, and is designed to operate on each fiber completely sep-

arately from the other. Figure 1 is a block diagram of the complete interface card.



The ODL's are the optical data link devices which connect to the optical fiber cabling.

The Taxi chips provide the 150 Mbps interface to the data streams between the ODL's and the receive and transmit engines.

FIGURE 1 **TCA-100 ATM Interface**

The TURBOchannel interface implements the TURBOchannel slave interface protocol. The TURBOchannel ID PROM contains information about the card that is used by the kernel during system boot and TURBOchannel initialization. The Receive and Transmit Queues support no-wait state access while the ID PROM and Control/Status registers require 6 wait cycles. This means that access to the control or status information needs to be kept to a minimum, as we shall see later.

The Control register provides the following functions:

- Receive cell count interrupt enable.

- Receive end of message interrupt enable.
- Receive timer enable.
- Transmit cell count interrupt enable.
- Receive engine enable.
- Transmit engine enable.
- Receive path reset (engine and cell counter reset, queue emptied).
- Transmit path reset (engine and queue reset, queue emptied).

These functions allow interrupts to be set on various events to allow interrupt driven drivers to control the frequency of ATM interface interrupts. I did not make use of the interrupt facility provided by the interface. The response time for processing interrupts would have added a large overhead during high speed data transmission, which would have slowed down the rate of data transmission. Applications that make use of the interface could well use these interrupts in cases where speed is not vitally important.

The Status register provides bits that are set in the event of the following conditions:

- Receive cell count interrupt.
- Receive End of Message interrupt.
- Receive timer interrupt.
- Transmit cell count interrupt.
- Receive cell lost.
- Receive no carrier.

Polling this register is required to determine the source of an interrupt, or can be used to monitor the given conditions when no interrupts are set. Cell loss can be caused by incomplete packets on the network or by a receive queue overflow.

The receive cell count interrupt is generated using a pair of registers, a receive count register and a receive count comparator register. This is used to set up an interrupt when a given number of cells have been received and is usually used in conjunction with the receive timer. The receive timer is reset to the value of the receive timer register whenever the receive queue becomes non-empty. These two systems of generating interrupts can be used together to prevent interrupts happening too often when the data rate is high, but often enough when the data rate is low.

A transmit cell counter and transmit cell count comparator pair of registers work in a similar way as the receive count and comparator pair of registers. No transmit cell timer exists, since this would probably not be a useful function.

The Receive Queue is 32 bits wide and 4096 words deep. This is enough storage to buffer 292 cells. The queue can be filled from the network in 800 microseconds if not drained. This size buffer is not enough to prevent overruns during reception of VMTP messages. Overrun behavior is to drop further incoming packets. Since 292 cells is at most 14016 bytes and more typically 12848 bytes in the case of the SAR cell format, this is less than a maximum sized VMTP message.

This places very tight timing constraints on a receiving machine during a burst of network activity. This type of burst can be created by

VMTP if a single VMTP send call is made with the maximum message size of 16 kilobytes.

Having overruns and cell loss causes severe performance degradation. This degradation is caused by two factors. First, packet retransmission, if driven by time-outs, causes long delays before the missing packets are detected. Second, standard packet reassembly techniques drop packets that have missing cells, and request complete packet retransmission. This behavior is the result of the difficulty of packet reassembly at high cell rates.

The receive engine aligns the cell on a word boundary for insertion into the receive FIFO and computes the CRC, if required. Bad cells are discarded if shorter than 5 bytes, or inserted into the FIFO and flagged as bad.

The Transmit Queue is 32 bits wide and 512 words deep. This provides enough storage for 36 cells. The network can drain this queue in 100 microseconds if no cells are added. Overrun behavior on writing to the transmit FIFO is to block the write until space is available in the FIFO to take the data. In theory, this allows one to copy cells into the FIFO as fast as the processor can copy the data. With the original cards, this occasionally caused a TURBOchannel time-out, which crashes the machine. The TURBOchannel can be stalled for 256 TURBOchannel cycles, which should allow the card to transmit many words from the FIFO. What was being experienced, as mentioned before, was the adapter card failing to release control of the TURBOchannel bus, thus blocking the CPU from completing a write operation. After the allowed 256 bus cycles had elapsed, a watchdog timer on the bus raised an exception, causing the machine to crash due to

the bus time-out. The problem was reported to FORE systems, which replaced the cards with components from different manufacturers, but carrying the same version number. The new cards were not visibly different except for the use of components from different manufacturers. There was insufficient time to test the new cards to see if the problem had been solved, but as was claimed earlier, the performance results were not affected.

The transmit engine starts transmission whenever there is at least one complete cell in the transmit FIFO. If required, it also computes the cell CRC field and inserts it into the cell before transmission.

Cell Interface Format

The receive interface supplies the cells as 14 words, which is 3 bytes more than the cell size. Figure 2 shows the cell format as presented by the receive interface:

TCA-100 Cell Format

byte0	byte1	byte2	byte3	Host Byte Order
byte4	Pad1	byte5	byte6	Host Byte Order
byte10	byte9	byte8	byte7	Network Byte Order
byte14	byte13	byte12	byte11	Network Byte Order
byte18	byte17	byte16	byte15	Network Byte Order
byte22	byte21	byte20	byte19	Network Byte Order
byte26	byte25	byte24	byte23	Network Byte Order
byte30	byte29	byte28	byte27	Network Byte Order
byte34	byte33	byte32	byte31	Network Byte Order
byte38	byte37	byte36	byte35	Network Byte Order
byte42	byte41	byte40	byte39	Network Byte Order
byte46	byte45	byte44	byte43	Network Byte Order
byte50	byte49	byte48	byte47	Network Byte Order
byte51	byte52	Pad2		Host Byte Order

FIGURE 2 **TCA-100 Cell Format**

Note the byte ordering of the various words of the cell. If the SAR cell format is examined (see SAR format diagram in Figure 3, on page 51) the ordering of the SAR payload is left in network order while the cell header, the SAR header and the SAR trailer words are supplied in host byte order. This prevents the need for the ATM Adaptation layer to worry about network byte ordering. The SAR payload however can-

not be byte swapped, since it is passed up the protocol stack to higher layers which expect network byte ordering.

Pad1 is used to give the computed CRC for the cell header (byte 0 to byte 4). It is zero if the header CRC was correct. In the event of a single bit error, its value can be used to correct the bit error without recalculating the CRC for the header. The cell header CRC polynomial is $x^8 + x^2 + x + 1$ XORed with 01010101. Pad 1 need only be checked if single bit error correction is to be attempted. This is because Pad2 contains a bit which indicates whether the CRC was correct or not.

Bit 0 of Pad2 indicates a framing error. Bit 1 indicates if the cell header CRC is correct. Bit 2 indicates if the payload CRC is correct. The payload CRC is the second CRC that is optionally computed by the transmit engine if a Class 3/4 ATM Adaption Layer (AAL) protocol is being used. This is the typical SAR cell format. A check of these 3 bits is sufficient to determine if the cell is good or bad.

Bits 3 to 5 are unused and set to 0.

The upper 10 bits of Pad2 contain the computed payload CRC, which can be used to correct single bit errors. This CRC is computed using the CRC polynomial $x^{10} + x^9 + x^5 + x^4 + x + 1$. These bits contain 0 if the payload CRC is correct.

The transmit interface accepts cells in almost exactly the same format as the receive interface delivers. The only difference is in the use of the Pad1 byte and the Pad2 bytes. Pad1 bit 0 is used to indicate if the header CRC should be computed by the transmit engine and placed in byte4 or if byte4 should be sent unchanged. Normally the header CRC is computed by the transmit engine and inserted. The main use of suppressing the computation of the CRC is to send cells with incorrect

CRC's to test the error handling and error recovery code at the receive interface.

Bit 1 is used to indicate whether the Class 3/4 AAL payload CRC should be computed and placed in the last 10 bits of the cell. If the SAR cell format is being used, the suppression of automatic payload CRC calculation can be used to test error detection and possible recovery at the receiving engine. If the cell is not a Class 3/4 AAL cell, this bit is turned off to allow cell data to be sent in the last 10 bits of the cell.

Currently Pad2 is unused by the transmit interface.

The address space of the ATM interface adapter card looks like the following:

Table 1: TCA-100 Address Memory Map

Address bits<17..16> (HEX)	Device
00	TURBOchannel ID PROM
01	Registers
10	Receive queue
11	Transmit queue

The ID PROM is a byte wide device, all other devices are word wide.

Access to the receive and transmit queues is through memory mapped interfaces that have the following structure:

Table 2: TCA-100 Receive and Transmit Queue Access

Address bits<5..2> (HEX)	Word
0	Header
1	Payload
2	Payload
3	Payload
4	Payload
5	Payload
6	Payload
7	Payload
8	Payload
9	Payload
A	Payload
B	Payload
C	Payload
D	Trailer
E	Unused
F	Unused

The order of writing a cell into this interface is very important. First the header word must be written, followed by all the payload words, followed by the trailer. The writing of the trailer notifies the transmit engine that a complete cell is ready in the case of the transmit interface. The words 0x1 through 0xC are functionally equivalent. In other words, writing or reading any

of the addresses accesses the next payload word. This allows the device driver to avoid address calculation during the loading of the payload. It does mean that the payload has to be written or read in the correct order.

Violating the order causes a deformed cell to be sent.

A cell can be discarded from the receive queue by writing a *byte* to the receive queue. This can be used to discard malformed cells that may arise or cells that are not wanted.

The Segmentation And Reassembly (SAR) cell format (shown in Figure 3) was used to allow larger packets to be segmented and then reassembled at the receiving machine.

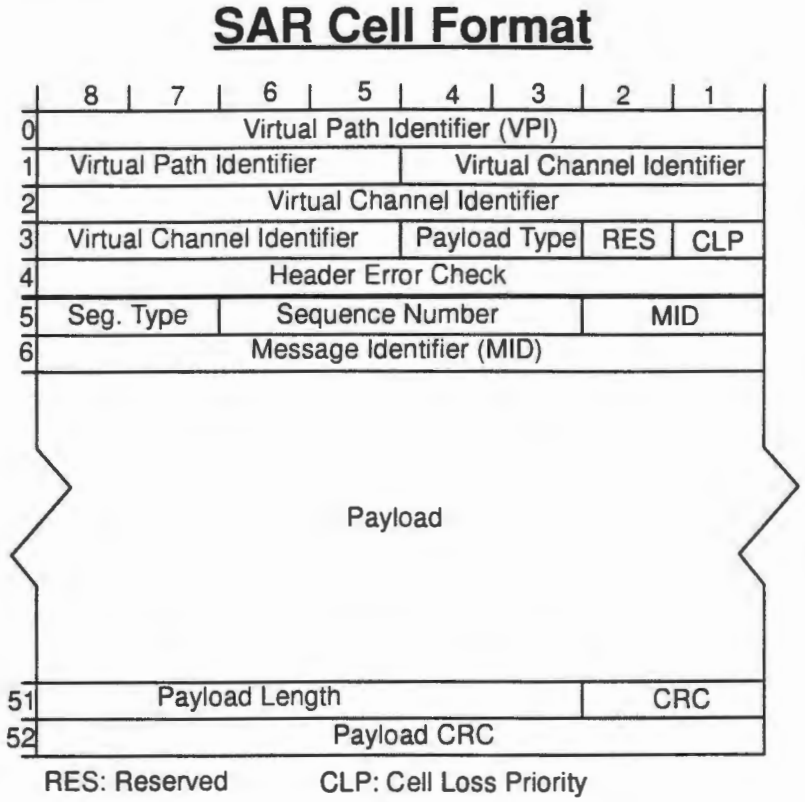


FIGURE 3 SAR Cell Format

The SAR format allows for a message to contain up to 16 cells (4 bit sequence number) which allows a 704 byte message. Ten bits are used for the message identifier. For the purpose of sending VMTP packets over ATM, I modified the sequence number field by adding a bit I removed from the message identifier field. This allows 1024 byte (1 kilobyte) packets to be segmented into cells that form a single ATM message. It provided a simpler logical mapping of 1 kilobyte packets to messages. The change halves the receive window size for message

identifiers and results in a reduction of the number of messages that can be outstanding at any one time for sliding window protocols. This is due to the halving of the receive window size that is possible using the message identifier field.

The reduction in possible receive window size from $2^{(\text{MID bits} - 1)}$ which is 2^9 (512 cells) to 256 cells. This is less than the receive queue size which could result in the receive queue being underutilized if messages are being received on only one virtual circuit. This inefficiency would not be noticed in a normal operating environment with more than one virtual circuit in use at any one time.

This difference in the SAR protocol format made no difference to the measurements that I made because I was not using a sliding window protocol. Messages with missing cells were discarded and complete messages were resent based on requests from higher up in the protocol stack. This proved to increase performance on the relatively error free network I was using over doing the more complex cell reassembly management required to transmit requests for cell retransmission and to maintain a strict sliding window protocol.

Driver Design

The design of the sample driver programs supplied with the interface were discarded due to the poor performance they exhibited. The driver that I wrote was written at an application level, using direct memory mapping of the ATM interface. All measurements were thus subject to the normal operating system overhead of a Unix application program. All measurements were taken with no other users on the machine and no other unnecessary active applications. (Other than the

required X11 server, xterms, etc. No screen activity or mouse events were generated during any tests.)

The driver was constructed from the bottom up. Measurements were taken for each aspect of the driver's functionality. The cost of various memory configurations for cells, caching effects and the costs of various interface reads and writes were measured. The driver was then constructed in the following order, with detailed measurement and performance optimization at each level before going on to the next level.

- Fixed words were written from a CPU register to the driver to form cells. This eliminated any caching effects and allowed the writing of full cells to be optimized.
- Preformatted cells were written from the cache to the interface. This allowed the development of optimum code for reading cells from memory.
- Preformatted VMTP packets (68 byte header + (no data to 1 kilobyte data)) were split into cells with correct sequence numbers. The VMTP packets were read from cache to prevent the performance from being affected by cache misses.
- Real VMTP packets were sent for performance measurements. Measurements were taken at 1 byte increments to the VMTP packet size.

Significant performance increases were obtained by special casing every possible ATM cell size. This special casing consisted of a large case statement that wrote the required number of bytes of data into the cell followed by the correct amount of padding to fill the payload.

Since the payload for each cell is so small, the use of bcopy did not provide any performance benefit. The best performance was obtained by generating a Cell Header, SAR Header and a SAR trailer that assumed a full ATM cell payload. The SAR Header sequence number was updated on the fly as each cell of a packet was written to the interface. This was done by adding 0x00000200 to the first word of the cell. Remember that the sequence number now included the former most significant bit of the message identifier.

Two further special cases were generated. When the bytes remaining in the packet to be sent was less than a complete cell, the last cell was special cased to copy the correct amount of data and padding into the last cell of a message along with a hard-coded SAR trailer. The SAR trailer can be hard-coded since the only variable field is the 6 bits which indicate the payload length. The last special case was to handle VMTP packets with between 0 bytes and 20 bytes of data. These are VMTP packets that fit into 2 ATM cells. The reason for special casing these VMTP packets is the study of packet sizes [10] which indicated that a large percentage (70%) of VMTP packets contain no data outside the header. The reason for special casing packets with up to 20 bytes was to allow the efficient filling of the second ATM cell. The second optimization accounts for the much higher performance of small VMTP packets (less than 20 bytes) than for larger packets.

The DECstation 5000 architecture uses write-through CPU caches. In order to improve performance while using write-through caches, the processor to memory interface incorporates a write-ahead buffer. This buffer is enabled for all addresses which means that even though the addresses that are being used to map the ATM interface are in the non-

cached memory address range, our writes are subject to the write-ahead buffering.

The effect of write-ahead buffering is to require that the write-ahead buffer is flushed before and after any writes to the cell header or trailer words. This is because the write-ahead buffer insures that every write to a memory location is completed before another access to that location is attempted. It does not guarantee the order of writing different addresses back. If the order of the header and a body word are reversed, the TCA-100 will generate invalid cells and cell loss will result before the cell even reaches the network. The receiving interface may even be required to reset its receive engine and receive FIFO after such an event, resulting in even greater cell loss. The flush costs valuable cycles as it holds up the processor and the TURBOchannel bus.

Performance on a DECstation 5000/240 was limited to a total transfer rate into the interface of 134.6 megabits/second which yields a payload transfer rate of 111.7 megabits/second. This represents the maximum sustained transfer rate of cells from processor cache to the ATM interface.

Implementation

I ported VMTP from Ultrix 4.1 to Ultrix 4.2a. Changes to the organization of the code in the Ultrix kernel between these two versions made it necessary to rewrite the hooks in the kernel as well as make adjustments for other changes in the Ultrix code.

The area in the Ultrix code where the various protocols get hooked into the decoding of an ethernet packet had been changed. The changes required a detailed understanding of the flow of control

within the Ultrix kernel from the handling of network interrupts, the decoding of the various protocols and an understanding of the workings of the network protocol code. Several modifications were required to both the kernel and the network code. The ported VMTP code had then to be tested to ensure correct functionality and reliable operation. The code was tuned until it performed correctly and with enough reliability for the code to be put into everyday use, replacing the original Ultrix 4.1 implementation as the implementation used to run the V and V++ systems storage servers.

I also implemented the first two prototypes of a memory to memory communication system. The first was written in C on the V operating system kernel and the second prototype was written in C++ on the V++ system. The idea behind this implementation is to remove all the network protocol code from the kernel to reduce the kernel size and also allow user level programs to take a more active role in the network. The hooks supplied by the kernel provide the application level software the ability to determine exactly what functionality it requires. The conventional kernel with built-in network support software gives all applications the same support with the result that some applications are given unnecessary support which results in a cost to all applications by way of lost CPU cycles and other kernel resources such as memory buffers. This results in a lowering of the potential maximum throughput for the whole system. Other applications require more services than the kernel provides, often leading to a duplication of effort and similar waste of resources as above. In this second case, the application also has to implement protocol functions that are often similar to functions that are implemented in the kernel.

The argument in favor of removing all the support from the kernel and placing the functionality in a user level daemon process gives two major advantages to the system. The first is a decrease in kernel size and complexity, which is usually accompanied by an increase in reliability which comes about due to the decrease in complexity. The second is giving applications the ability to use the standard network functionality if they suit the application needs or to replace some or all the functionality with a different implementation.

This particular implementation was designed in two parts, a local machine implementation and a remote implementation. The local case is specialized so that no unnecessary work is performed on data destined for the same machine. The kernel provided a signaling mechanism and hooks to set up shared memory segments. In the case of the Paradigm multiprocessor hardware, the signal mechanism is provided by the hardware, thus removing the need for traps into the kernel to generate the signals and an additional saving. The hardware can be configured by the kernel to generate a signal when the last word of a cache line is written. In the Paradigm hardware, this action can also trigger the transmission of the cache line over a high speed network resulting in the delivery of the cache line into the second level cache of another machine.

The implementation from the application level, which I did, finds the model used by the Paradigm hardware and the DEC 5000 machine almost identical. The calls to the kernel to send signals can be removed on the Paradigm machine, although such a change will not make any difference. To use this network model, an application needs to include all the network protocol functionality that it requires. This is done in most applications by linking in a networking support library that

implements the standard network functionality. Applications with special requirements implement the needed functions. The use of a library and the general hooks that the kernel provides also gives applications the ability to use more than one set of protocols for different purposes.

I implemented the user level library, which supplies the applications the standard functionality that was provided by the kernel in the past, in a more flexible way. The model is connection-oriented, with channels established between two processes that need to communicate. In the local case, the channel is a shared memory segment with the signaling mechanism providing the trigger for the receiver. When a message is sent for the first time, the channel and signal handlers at each end of the channel are initialized. The data is then written into a slot (or cache line in the case of the Paradigm hardware) and a signal is sent on that slot. The receiving end receives the signal and reads the information out of the slot. The library provided the needed collision detection and packet ordering that applications may need.

In the case that the destination is on another machine, a user level network process acts as a local proxy for the remote process. Thus the application sees exactly the same model of the channel as in the local case. The proxy network driver receives the signal and packs the contents of the signaled slot into a network packet of whatever format is desired. On the Paradigm, if the hardware network support is used, the transmission is done by the hardware. On the receiving side, the network packet is received and accepted if no errors are detected. The packets contents are then put into a channel that connects the network driver (or remote proxy) to the final destination process. The driver

then signals on the slot and the receiving process cannot tell that the channel does not connect to a local process.

Another implementation that I did included a device driver and protocol layers for an ATM network. This required controlling the ATM network card at the register level as well as forming ATM packets of the correct format, doing the necessary packet fragmentation and reassembly to allow larger, more usual ethernet packets to be sent over the ATM network. I measured the ATM network extensively to analyze its potential speed on the DECstation 5000s that the ATM cards were designed for. The hardware consisted of a DECstation 5000/200 and a DECstation 5000/240 each fitted with an ATM card. The network was a point to point network without an ATM switch. Later implementation work included making the necessary changes to the device drivers supplied with the cards and the Ultrix kernel to support VMTP over the ATM network. This was not completed due to the poor performance, based on the analysis I performed, relative to the cost of the hardware, which reduced the value of completing the project. As will be presented later, the cards have a high potential throughput, but the potential is largely reduced by software and memory speed considerations. The early evaluation did however provide a lot of insight into design of high speed networks and their protocols.

The ATM card provided a memory mapped image of the hardware. This map consists of a number of hardware control registers that provide information about the state of the interface, an area into which outgoing packets are written and an area from which received packets can be read. The packet interfaces consist of a number of areas which are required to be read or written in specific orders in order for the

cards FIFOs to operate correctly. This causes a major constraint in the way the data reads and writes can be optimized.

Other work involved doing extensive work on the V system storage server. This involved a lot of client and server implementation and debugging of the servers, the Ultrix VMTP network code and of the VMTP protocol implementation. Considerable effort had to be expended in an effort to get long term stability in the Ultrix implementation. This instability was the result of the complexity of the VMTP protocol implementation. This gives an insight into the difficulty of providing a reliable implementation of a real protocol of any reasonable complexity. The VMTP protocol starts out very simple in design but becomes very complex as all the necessary special cases are taken care of. These cases are largely concerned with issues such as errors and special casing various situations in order to get good performance. The implementation was greatly complicated by the lack of threads in the Ultrix kernel which results in many difficulties, particularly in the area of timer and error handling.

ATM Performance Measurements

Once the driver development was completed, measurements were taken to understand the performance of the ATM SAR protocol using various sized packets. Packet sizes were incremented in units of single bytes. The smallest amount of data sent is recorded in the following graphs as 0 bytes. This would be a message consisting of a single VMTP packet header without any data segment, 68 bytes of ATM payload data. The largest amount of data tested was a VMTP packet containing 1024 bytes of data, this being a total of 1092 bytes of ATM pay-

load data. All data amounts recorded between these limits were derived in the same way.

This method of data size computation is consistent with typical uses of an ATM network, which is to transport higher level protocol packets. The FORE Systems TCA-100 interface card is packaged with a driver that allows the card to be accessed transparently as a TCP/IP network. This use of ATM networks will continue due to the limitations of the simple SAR type protocols. This protocol simplicity is a requirement imposed by the very small cell sizes that are used on ATM networks. The approach is similar and consistent with the practice of developing reliable delivery protocols using lower level protocols that may not be reliable, such as UDP.

This layered approach to protocols allows highly tuned lower levels that provide good performance while allowing the higher layers to add richer communications semantics to be built into the overall protocol stack. It is an approach that allows the same lower level protocols and semantics to be reused repeatedly in the development of all the rich application level protocol semantics that are required. This reuse of lower level code simplifies porting of the protocol stack to new architectures and new network technologies. This has significant real world advantages to software development companies. A final point in support of this approach is that applications requiring only simple protocol semantics are never required to pay a penalty due to support for more complex semantics.

The graph in Figure 4 shows the relationship between the amount of data sent in a packet and the percentage of time spent copying the data. This is mostly the copying of data from the data source to the

ATM interface. The segmentation algorithm that I developed split the data buffer as it copied to the interface, thereby eliminating unnecessary copies from being performed on the data.

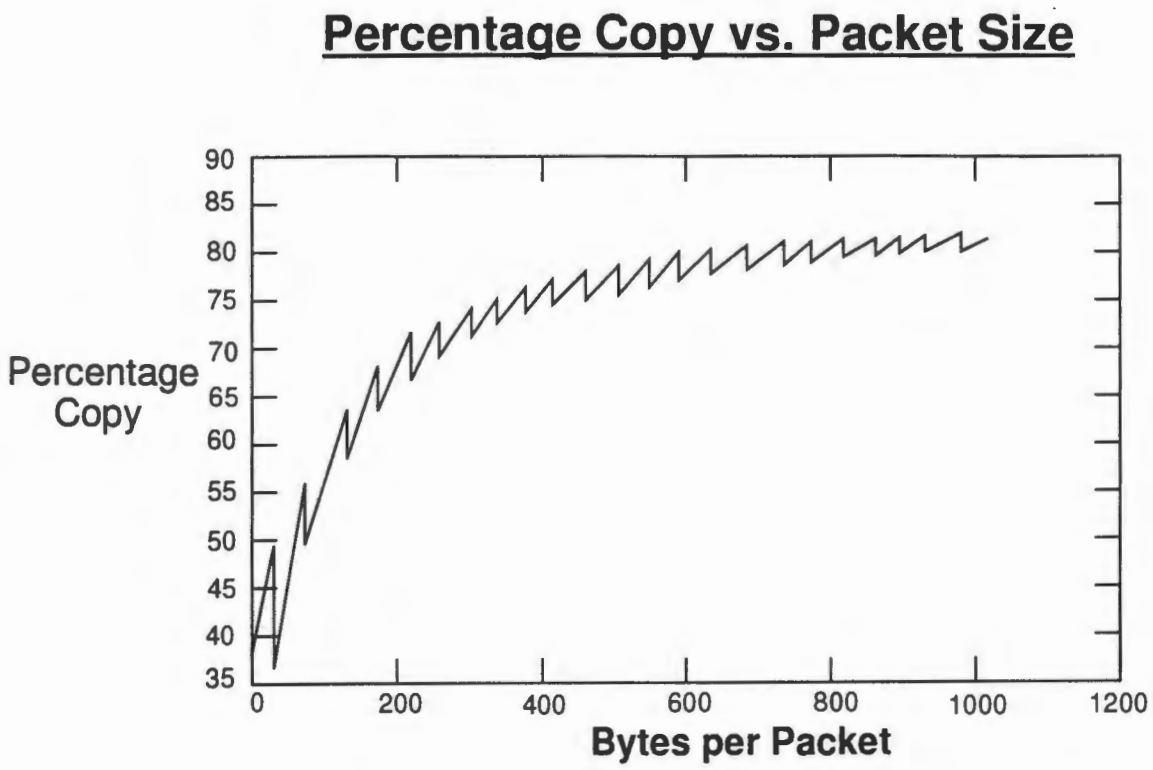


FIGURE 4 **Percentage Copy vs. Packet Size**

The lowest copy cost relative to total transmission cost is incurred with packets containing 21 bytes of data. With VMTP packets which contain this much data, the total payload data size for the ATM cells is 89 bytes, which completely fills 2 ATM cells and consumes a single byte from a 3rd cell. This is the first data size that does not follow the highly optimized code path that was designed to be efficient with empty and near empty data segments in the VMTP packets. All packets containing more data than this follow the same code path.

The repeating saw tooth shape in Figure 4 is due to the repeated inclusion of new ATM cells into the chain of cells that are required to carry all the data. The peaks occur at 44 byte intervals. The low end of the peak coincides with the amount of data that causes the last ATM cell of the packet to contain only a single byte of data. Having a cell contain only a single byte of data still requires a cell header, a SAR header and a SAR trailer to be copied to the interface. The overhead of this copy, the writing of the filler data to keep the cell size constant and the cost of calculating the SAR header and trailer all count as overhead and are thus not counted as part of the copy cost.

The high end of each peak coincides with the amount of data required to completely fill all ATM cells that are being sent. This has a relatively higher cost than a cell containing less data since the fixed cost overhead of sending each cell remains constant while the cost of copying data into the cell increases as more data is required to fill the cell.

The overall curvature of the graph is explained by the fixed cost of generating the initial headers for the cells as well as the message identifier. This cost is slowly amortized as more data is sent. The percentage of the total cost attributed to copying then slowly increases. This trend stabilizes at large packet sizes at over 80% of the total cost of sending a packet. The dominant cost of sending large messages using ATM is the cost of copying the data to the interface. This cost consists of the memory system access times, the CPU processing costs, and the necessary bus access times associated with the data transfers.

Figure 5 shows the relationship between the total network throughput and the size of the packets that are sent. These measurements were taken using a series of preformatted VMTP packets that were pre-gen-

erated and then repeatedly sent over the ATM network. This method of data generation was used to eliminate the cost of VMTP packet formation from the total cost of sending ATM cells over the network. The graph contains values obtained for both the DECstation 5000/200 and a DECstation 5000/240. These give a comparison of the effect of different CPU speeds in ATM network throughput.

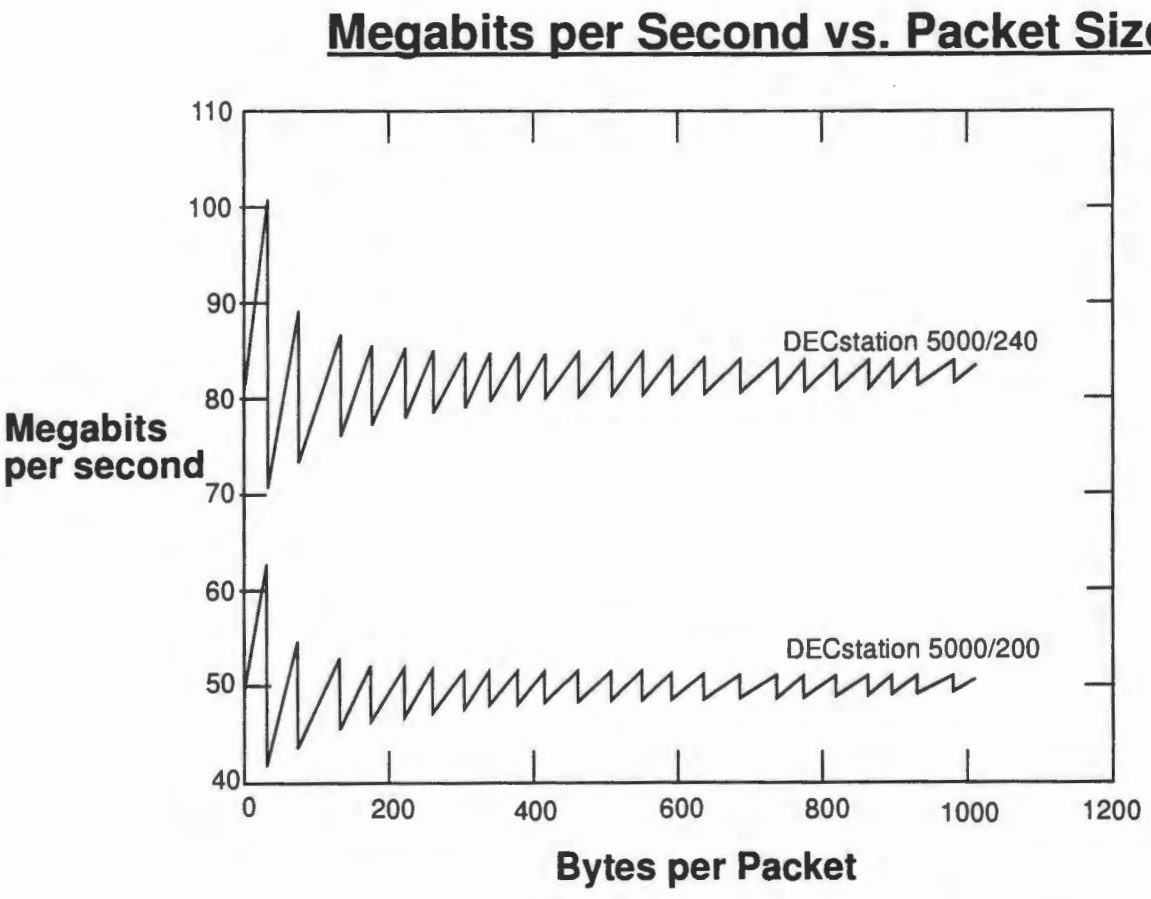


FIGURE 5 **Megabits per Second vs. Packet Size**

It should be noted that under some situations, it was possible to get worse performance using the DECstation 5000/240 than the DECstation 5000/200. With certain configurations of the driver implementa-

tion, the CPU experienced penalties when accessing the TURBOchannel bus that could not be recreated on the DECstation 5000/200. The cause of this seemed to be a mismatch between the 33MHz CPU speed and the 25MHz bus speed. The periodicity of the TURBOchannel bus accesses governed the severity of this problem. It also only occurred with a very small range of access timings which seemed to indicate interference with other TURBOchannel activity, for example memory refresh, screen buffer transfers or some other periodic bus activity. The measurements given here are based on driver implementations that did not trigger this behavior.

The two machines differ in the total throughput. This is primarily the result of the processor speed difference as well as the difference in cache speeds. The rest of this discussion will be about the DECstation 5000/240 performance curve.

The best performance was obtained for the 2 ATM cell case, which corresponds to 20 VMTP data segment bytes for a total of 88 ATM payload bytes. This particular message size results in the most highly optimized code path in the device driver being used. Smaller packet sizes show lower throughput because partially filled ATM cells have to be padded up to the maximum cell size. The overhead of copying this non-payload data reduces the overall bus bandwidth available to real data.

The lowest performance is shown for a total of 89 bytes of data. This size requires the general code path to be used and also requires a third, almost empty cell to be sent. The addition of a third cell uses a total of a third of the total network bandwidth and almost a third of the bus bandwidth with practically no return in the amount of data

sent. The reason that the bus bandwidth waste is not a full third is that the filler data does not have to be fetched from memory. Thus it only crosses the TURBOchannel bus once on its way to the interface.

The convergence of the graph in Figure 5 towards 80 megabits/second is due to the amortisation of the fixed costs over an ever greater number of ATM cells and the reduction in the percentage of bandwidth wasted not carrying data in the final ATM cell of each packet. This is because only one ATM cell of each packet contains less than 44 bytes of data. In the case of 89 bytes of data to be sent, 43 bytes of filler are sent, which constitutes 48.31% of the data size and a total of 32.58% of the data that is transmitted over the network. If on the other hand the total data size is 1057 bytes, the 43 bytes of filler constitutes only 4.07% of the data size and is only 3.91% of the total data transmitted over the network.

This shows that correctly sized messages can make a significant difference to performance, particularly when small messages are being sent. This has implications for many applications such as video or audio data over ATM. The use of full ATM cells results in more efficient utilization of the network bandwidth, since complete cells are sent even with partially filled cells.

The graph in Figure 6 shows the relationship between the number of packets that can be sent per second and the packet size. It shows the number of ATM cells and the number of VMTP packets. The two lines

on the graph are related, the upper line shows the number of ATM cells that are sent corresponding to the size of the VMTP packet.

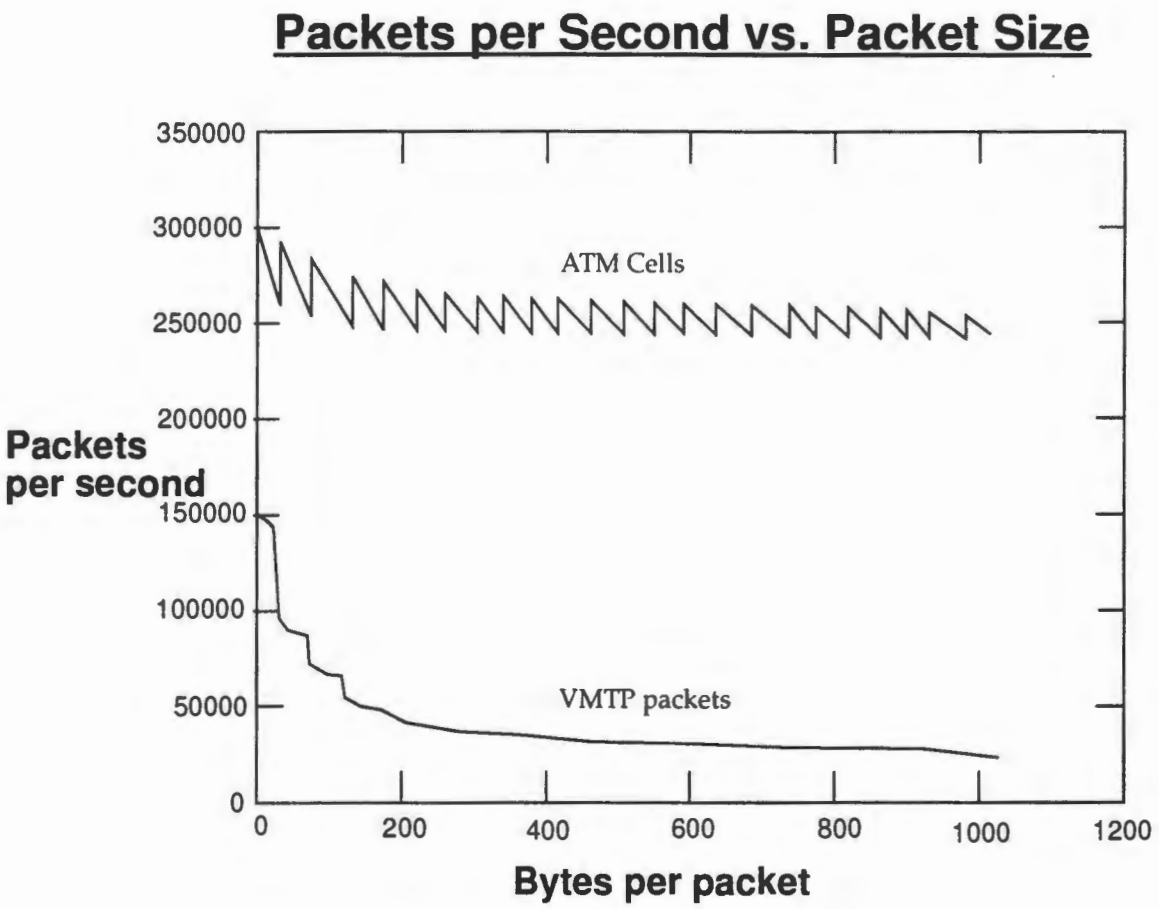


FIGURE 6 Packets per Second vs. Packet Size

The initial high number of VMTP packets per second is in the size range that follows the most highly optimized code path. After the initial drop in the packet rate, the VMTP packet rate falls in a series of steps. At each step, more ATM cells are required to send the data. The size of the steps decreases as the incremental cost of adding new cells becomes a smaller percentage of the total.

The ATM cell rate graph has a saw tooth appearance as it had in the preceding graphs. There is a slow downwards trend as more cells are used to send the data. The peaks occur at the points where a new ATM cell is added for each message. This causes a larger number of packets to be sent for each message with the smallest increase in overhead of copying data to the cells.

The low points on the graph come at the point where each cell is being filled completely, thus incurring the highest copy cost overhead. The cost of copying into a nearly empty ATM cell is lower than copying into a full cell since the filler bytes cross the bus only once, while the data bytes have to cross the bus twice.

Figure 7 shows the breakdown of the total time to send 100,000 packets of each size from 0 bytes of segment data which is 68 bytes total to 1024 bytes of data which is a total of 1092 bytes.

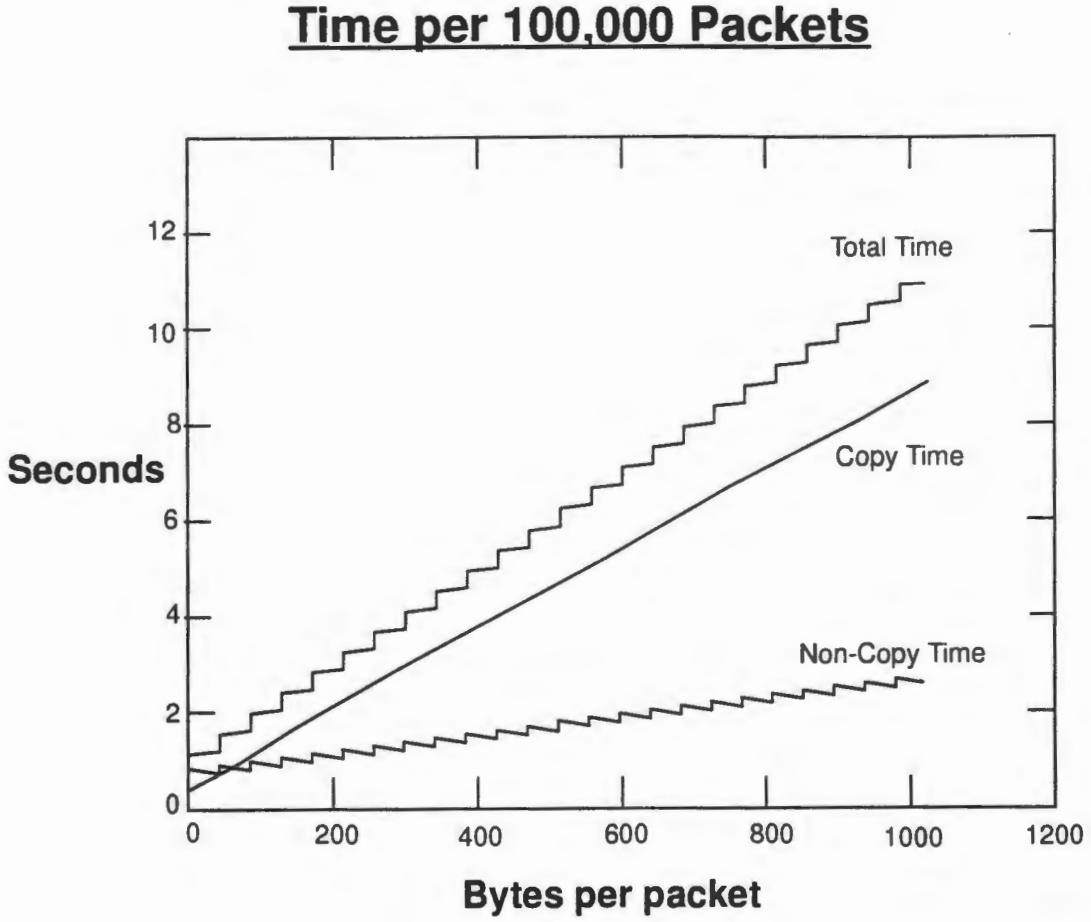


FIGURE 7 **Time per 100,000 Packets**

The total time line illustrates the amount of time taken to send the 100,000 packets of the given size including all transmission costs. This time is made up of the sum of the two lower lines.

The copy cost is as expected, a linear increase as the amount of data to be copied increases. It is again clear that the copy cost dominates the total cost of sending the packets. This would indicate that the biggest

gain in increasing performance would come from improving the copy performance. The biggest bottleneck comes from the poor use of the TURBOchannel bus. With the given interface, each word that is written to the interface can only cross the bus after TURBOchannel bus arbitration has taken place. This places a significant overhead on each write. Both direct memory access (DMA) and cache reads make use of burst transfers over the bus. With a burst transfer, the cost of bus acquisition is amortized over a large number of word transfers over the bus.

The non-copy time graph in Figure 7 shows the overhead of setting up the packets, updating the headers and filling the end of the last ATM cell with filler bytes. The latter time is what accounts for the saw tooth shape of the graph. As the last cell fills, the amount of filler data that needs to be copied diminishes. The linear overall trend of the line is expected, since a fixed increase in overhead is incurred with the addition of each ATM cell.

VMTP Measurements

The VMTP measurements took place between three different types of DECstation. These were the DECstation 3100s, DECstation 5000/200s and the DECstation 5000/240s, all running the Ultrix operating system. The testing was done using a lightly loaded 10 megabit/second ethernet network. Tests that showed results affected by sudden network loads were discarded. The remaining measurements were averaged to reduce the influence of random events on the measurements.

The design of the VMTP protocol has a large impact on its maximum throughput. The protocol is based on a request-reply communication system. Requests send data to a server and the protocol waits for the reply before the next request can be sent. This naturally limits the throughput by relating it to the network latency. The longer the latency, the less time is actually spent sending data. This behavior is not quite as bad as it seems, because VMTP allows streams of packets representing a message to be sent without individual acknowledgments on each part of the message.

Retransmission requests are not done blindly: messages that are partially received have the received parts accepted and a request is made for the retransmission of the missing parts of the message. This results in a substantial improvement in protocol efficiency on unreliable networks when large messages are being exchanged. The selective retransmission is implemented at a lower granularity than message size. In the current implementation this granularity is 512 bytes (half of one ethernet packet.) The selective retransmission is achieved by using a bitmap to indicate the missing portions of a message, which are then repackaged and resent. Since the amount of data that is sent in each VMTP packet over ethernet is 2 VMTP blocks of 512 bytes, it is

sometimes necessary to repackage the data blocks. This saves the amount of retransmission that is required on an error prone network.

The graphs that follow are based on the performance experiments that were performed on the Ultrix implementations of VMTP. Some of the work utilized the VMTP implementation that I ported, while the remainder utilized the code that I was involved in debugging and modifying.

Figure 8 shows the data rate that is achieved by a single client sending requests of the indicated size to a remote server. The server replies with an acknowledgment as soon as the packet is received correctly. The acknowledgment could potentially contain a reply value. In the test setup, any data returned from the server is placed in the data space within the VMTP packet header. Such data would have to fit

within the 20 bytes provided inside the reply header and could possibly use the response code.

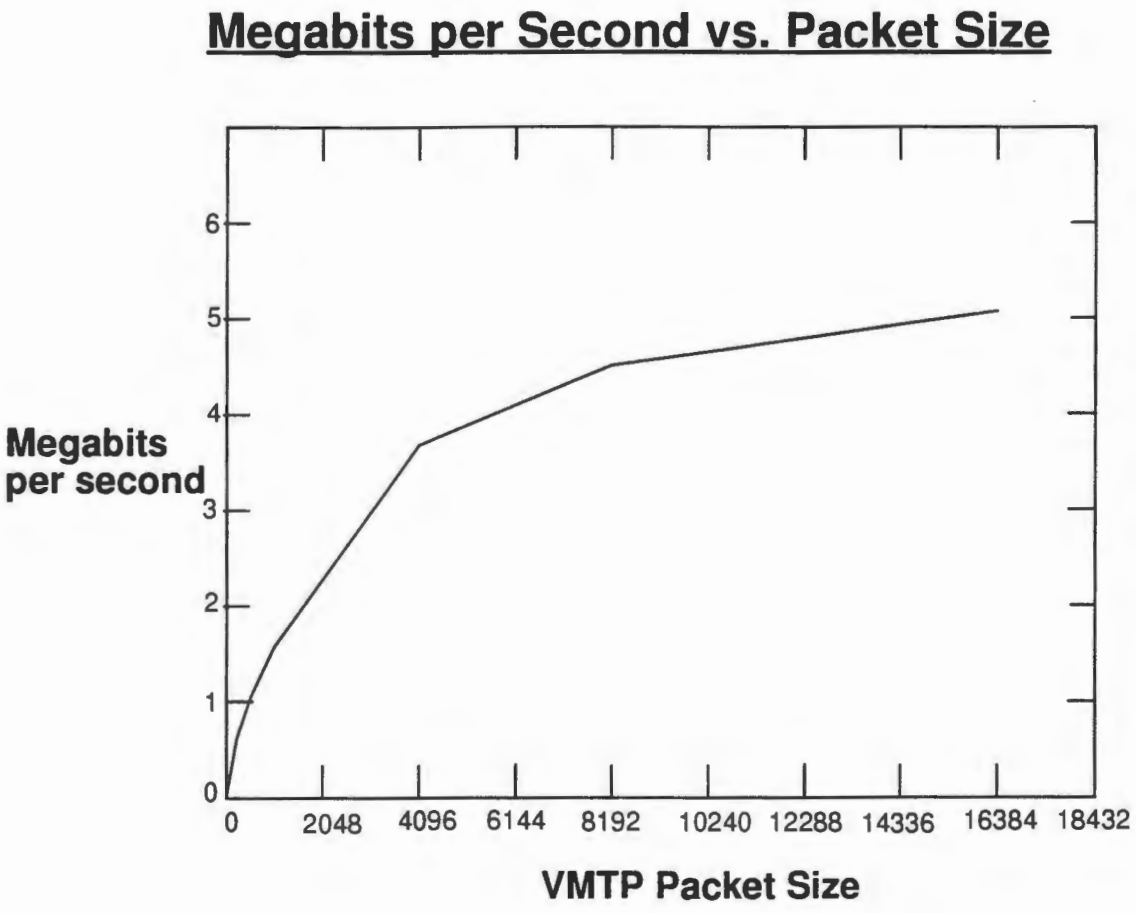


FIGURE 8 **Megabits per Second vs. Packet Size**

Figure 8 indicates the expected increase in throughput as the size of the packets increases. The major factor in causing this trend is the amortisation of the request-response latency over a greater number of payload bytes. A secondary effect is the increased percentage of data relative to packet header information that is being sent. This again reduces the amount of bandwidth that is wasted on non-payload data transfer.

The first effect mentioned above, that of latency, is most pronounced for small packet sizes. The measured latency for a VMTP packet with all request data placed within the 12 bytes provided for user data in the request packet header is 15.39 milliseconds. The time taken to transmit the entire header of 68 bytes is 0.0544 milliseconds. The latency for a VMTP message of 16 kilobytes is 139.776 milliseconds. (These measurements exclude time possibly spent through media contention.) The first case spends a total of 0.1088 milliseconds sending both the request and the reply, which is a total of 0.71% of the total transaction time. When sending a large request, 90.09% of the transmission time is used to transmit user data.

This difference accounts for the rapid increase in throughput as packet size increases. The incremental advantage of increasing the packet size decreases as the packet size increases. This causes the graph to flatten out rather quickly. The fact that the graph still has an upwards slope as the maximum message size for a VMTP packet is approached is an indication that the network has not reached its data limit. As the limit of the network capacity is reached the graph would become horizontal.

Saturating the network with a single client utilizing a synchronous request-reply protocol is not possible. This is because the very nature of a synchronous request-reply protocol makes it self limiting in the amount of data that it can send from a single client. This is the result of the client having to wait for a response before attempting to send the next request. The client is then throttled back as contention is encountered on the network.

This behavior does not exist with all request-reply protocols, only those that offer or require synchronous sending of the data. In other words, if the call blocks until a reply is available, the client is subject to automatic flow control. If this is not the case, the client can cause contention by sending many requests without waiting for replies. The outgoing requests can slow the responses to previous requests, allowing a single client to degrade system performance. The advantage of such a system is that it allows the client to make full use of the CPU and make better use of the network if it can send concurrent requests.

The graph that follows in Figure 9 shows the effect of multiple clients on the same machine all sending the largest requests to a single server which is returning the largest possible single message reply.

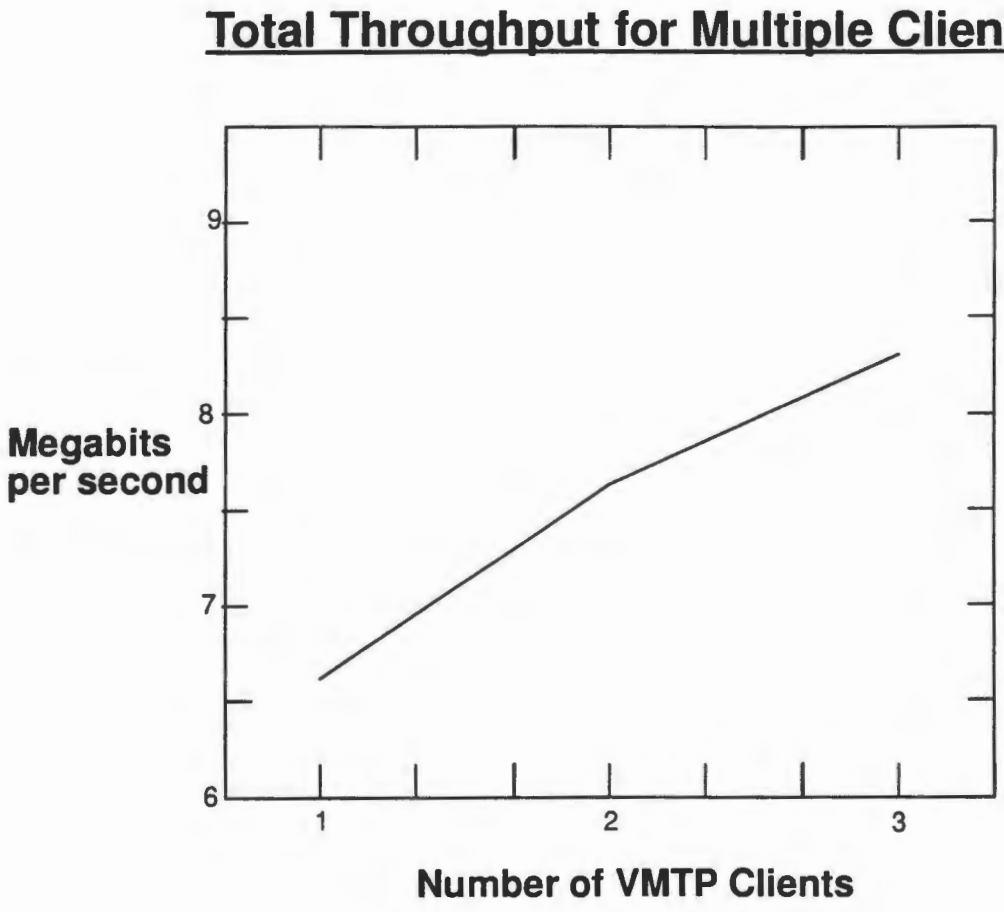


FIGURE 9 **Total Throughput for Multiple Clients**

The fact that both the reply and the request contain 16 kilobytes of segment data results in higher throughput than recorded on the previous graph when only the client sent large requests and the replies were small. As expected, the data rate increases as the number of clients on the machine increase. The increase is not uniform, because each additional client adds a smaller amount to the total. This test was limited

to only 3 clients due to some implementation problems at the time the measurements were made. The kernel on the client machine would begin to experience severe problems if 4 clients were used (reported throughput of 109 megabits/second) and would consistently crash when 5 clients were used. My speculation is that the buffer management in the kernel could not handle the large amounts of data that needed to be stored and managed within the kernel.

If more clients could have been run, the graph would have flattened out rapidly, since even with 3 clients, the network is operating at a very high throughput for a 10 megabit ethernet. The improvement in total throughput would have become very small with each additional client added. The utilization would not have exhibited the reduction in throughput that would have been expected if the clients were run on different hosts, or the servers were run on different hosts. In a multi-host situation, performance would have quickly begun to degrade as contention for the ethernet developed. The collisions caused by attempting to access the ethernet would have wasted an increasingly large amount of the available bandwidth as more hosts were added.

What the graph does show is that VMTP is capable of utilizing a very high percentage of the available ethernet bandwidth. This is particularly true when only a small number of hosts are involved in generating a large amount of the network traffic. Data having multiple destinations imposes no additional load on the network because of the possible use of VMTP multicast to groups of applications.

Figure 10 shows the result of having multiple clients on a single host. This time, however, each is sending the smallest possible VMTP mes-

sages and receiving the smallest possible replies. Both the Request and the Response packets consist of a 68 byte VMTP header containing up to 12 bytes of request data inside the header and 20 bytes of reply data in the reply header.

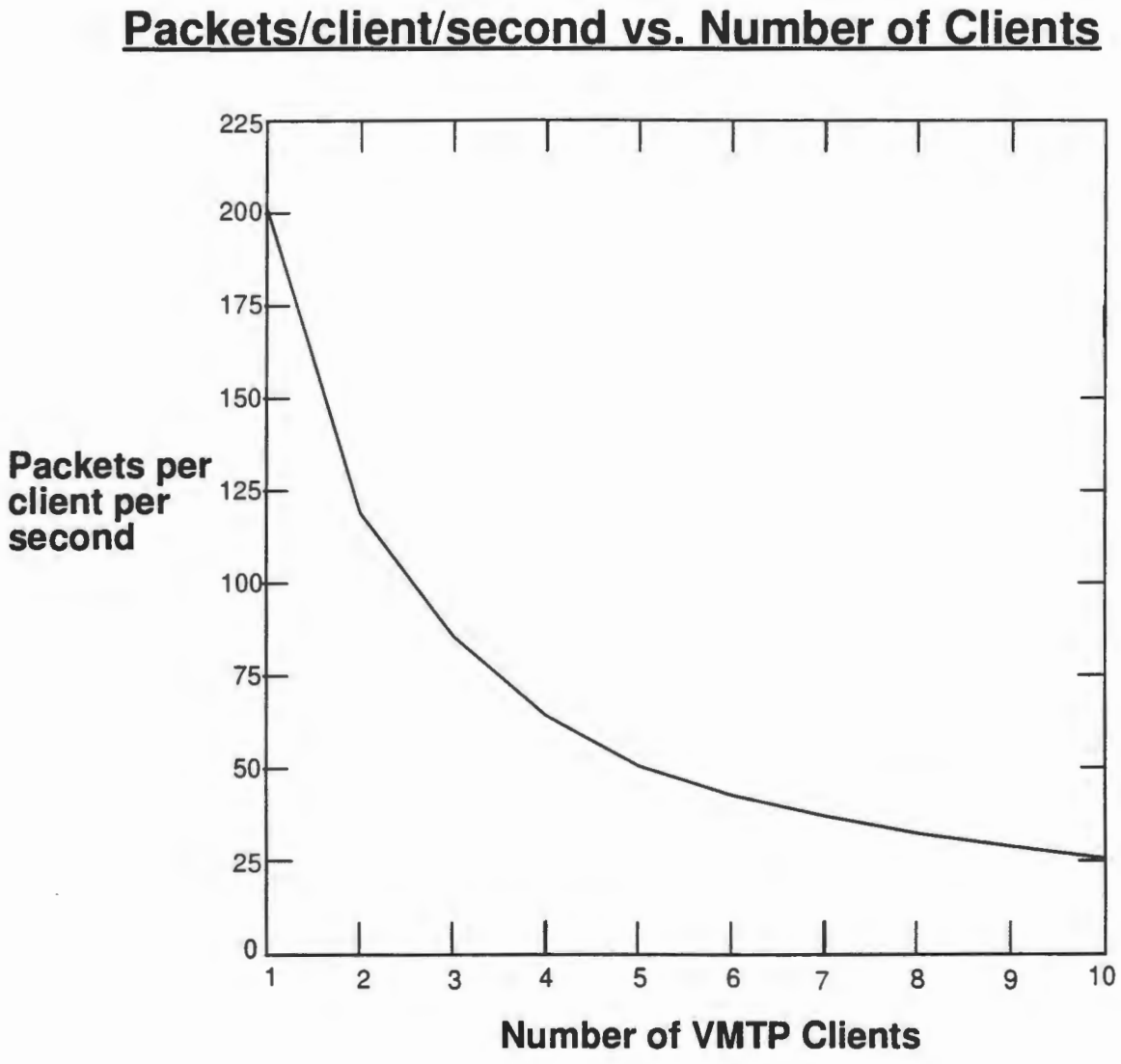


FIGURE 10 Packets/client/second vs. Number of Clients

The graph in Figure 10 shows the number of packets that can be sent in a one second period by each of the clients on the machine. As more clients try to send packets, the total number that each client is able to send decreases. This decrease is initially very rapid as more clients are added to a small number of clients, but the incremental change for each additional client gets smaller.

For very large numbers of clients, the graph will appear very flat, but monotonically decreasing. This is because the operating system and network resources are limited and for large numbers of clients, they are effectively sharing a fixed resource among many. The typical case on a machine is to have a small number of clients involved in communication at any one time.

Figure 11 shows the total number of packets being sent by the host used in the previous experiment.

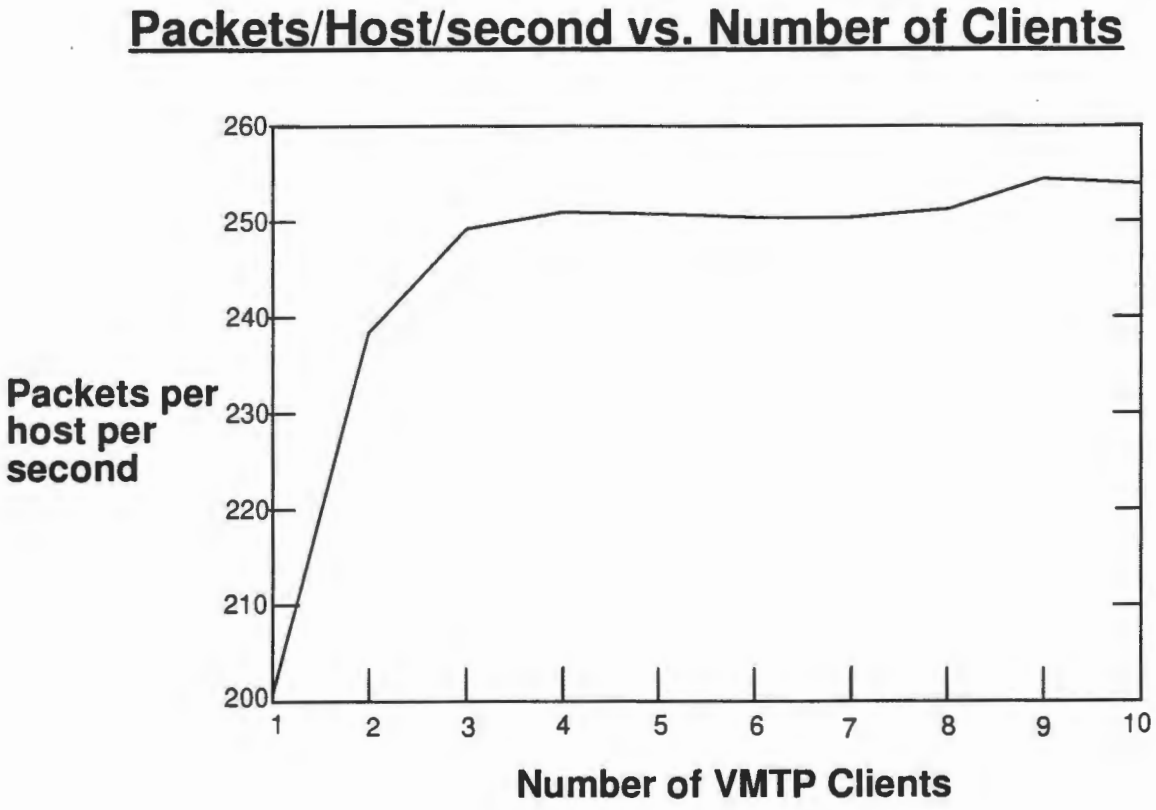


FIGURE 11 Packets/Host/Second vs. Number of Clients

This graph shows that the overall number of packets sent on the network increases quickly as more clients are sending simultaneously. The gain is due to the effects of latency on a single client. Multiple clients are able to utilize the time that other clients are waiting for a reply to previous requests.

This results in a very rapid increase in the number of requests that the machine can send as the number of clients increases. Very soon however, all the available time is consumed by operating system and net-

work protocol overhead, resulting in the total number of packets being sent to become very stable. As noted earlier, additional clients effectively just share a portion of this total

The previous two graphs show the limits that VMTP would place on individual clients, and on individual hosts for applications that needed high speed communication of small amounts of data. This size of message is common for interactive applications that deal with input such as mouse events. Distributed simulations would possibly also require the ability to send large numbers of very small packets.

Network traffic is dominated by very small packets and by very large packets with relatively few packets of intermediate size. The performance using small packets is very important for any application that is not involved in the transfer of large amounts of data. This would be typical of most RPC applications that have only a few bytes of state that need to be transported from client to server, and equally modest response requirements.

Interactive applications experience a lot of network traffic with similar characteristics. Keystrokes, mouse events as mentioned earlier, screen updates and responses to user input are predominately small in size and large in volume. Applications involved in transfers of large amounts of data make use of predominately full data packets. Since activity is dominated by these two extremes, small and large packets are the most common on the network.

Figure 12 shows the effect of VMTP message size on the time taken to send a request and get a reply back. This particular test varied the size of the segment data sent in the request from a single client to a server. The server responded to each request with a reply containing no data

beyond the few bytes of user data that can be placed in the VMTP reply header. The time measurement includes the processing time of the kernel on both machines and the small amount of processing that the server performed. The server workload involved sending a VMTP reply as soon as it received a request, thus this load is negligible.

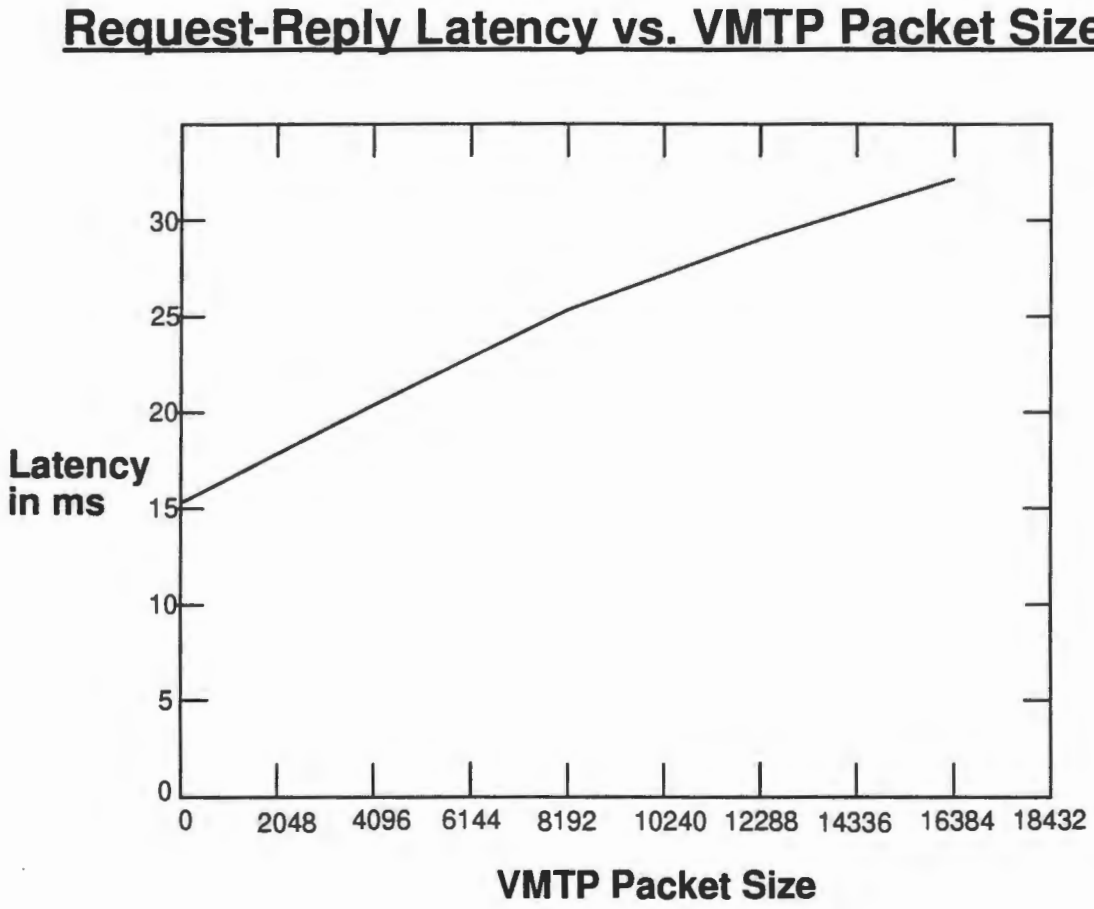


FIGURE 12 Request-Reply Latency vs. VMTP Packet Size

The latency for a transaction increases very nearly linearly as the packet size increases. This is driven by the increased copy costs in the networking protocol software as more data is sent, requiring increased copying and data storage. The cost of sending multiple small mes-

sages as done in earlier tests can be seen in this graph. Each transaction would take at least an additional 15 milliseconds. This would result in two transactions each involving the transmission of a single byte taking approximately the same time as a single transaction involving 16 kilobytes.

Clearly efficiency dictates sending as much data as possible in a single transaction, rather than using multiple transactions.

Chapter 5 Discussion and Conclusion

A fundamental problem in protocol design is the trade-off between protocol complexity and protocol speed. Functionality and added services add cost to the protocol in the form of added processing time required to handle messages. The protocol performance becomes a greater issue as network speed increases, because of the increasing rate at which data has to be handled by the receiving machine.

From my measurements of ATM networks, I found that protocol overhead becomes an issue at relatively low speeds of around 50 megabits/second. With bus bandwidths of around 12 megabytes/second, the cost of copying data from buffers to where it is required becomes a limiting factor. With the considerable work required to reorder packets and strip or reroute data as well as the error checking and retransmission requirements, current raw network performance outstrips the rate at which data can be processed by the receiving machine.

High network bandwidths can be effectively used for aggregate traffic on backbones. Local area networks typically do not require network bandwidths that are orders of magnitude larger than current local area network speeds. Live video network requirements depend on many details including; frame rate, image size, number of bits per pixel and compression ratio. An example of current live video feeds require only in the region of 700 kilobits/second [Silicon Graphics InPerson videoconferencing software]. Video conference use would require only some small multiple of that since only a limited number of participants would need to be transmitting at the same time. Even if 20 par-

ticipants required such a feed, the bandwidth would only be 14 megabits/second. This would represent less than 2% of a gigabit network.

If one considers that current systems can handle most *local* file transfer or file server access over 10 megabit ethernet, it appears that unless very network intensive applications are found, network requirements will be easy to meet with current technology for all but the largest backbone networks. On the other hand, we always find a way to utilize additional performance, whether it is memory, diskspace, CPU performance or bandwidth. We clearly do have a need in wide area networks for tremendous increases in network performance.

The problem that current machines are having trouble handling is the protocol work that the machine has to perform at even these modest network speeds. Main memory system speeds are a large part of the problem. As processor speeds have increased, memory systems have become multi-leveled, often with 2 or more caches [20]. Processor clock speeds are high enough that it has become inefficient to access main memory and large amounts of effort are devoted to designing cache systems that maximize cache hits.

Having a dedicated processor handle the protocol implementation of the network system seems at first like a good idea, but such an approach will only increase the cost of the network system. As general purpose processors get faster, the dedicated network processors will again become a bottleneck. Having dedicated hardware handling protocol also limits the types of service that can be offered. This leads to much more complex design decisions - how to anticipate future network usage and changes in protocols. Errors in the protocol implementation will be much harder to correct. An example of how net-

works may change is the proposed increase of the address space of the Internet protocol. Dedicated hardware could make such changes much more expensive.

If we want to use the main CPU to handle the protocol processing, then we have to decide how to overcome memory bandwidth limitations. One method that seems to have a lot of potential is for the network to share the processor cache at some level where the dual benefits of faster memory access for the network and faster access to the data by the processor pay off without causing expensive contention with the processor for access to the cache.

In a multilevel cache hierarchy with a small on-chip processor cache, a larger (and often slower) second level cache and a much larger and slower main memory, the second level cache would seem like a likely candidate[8]. The first level cache would be too small, even if access to it was possible and would compete too directly with the processor for first level cache access. Direct access to main memory would give us the current situation with all its limitations. Direct network access to the cache has to be handled very carefully, since it would be all to easy to create a situation in which useful data was being displaced by the incoming network data.

The level of extra hardware sophistication required to implement a network interface that communicates directly with the second level cache is not very large, as shown by David Cheriton in his work on the Paradigm machine [8]. Cache lines that are accessed by the network hardware should not be handled in exactly the same manner as regular cache lines.

In the Paradigm machine, cache lines can be set up to trigger a transfer of the cache line when it is updated. This ability allows the machine to set up routing of messages by having a destination cache line also set up to send when written to. This means that the routing of messages along a virtual circuit can take place without any CPU intervention once the mappings have been set up. The efficiency increases offered by this complete lack of copying data that has to be forwarded is considerable. As discussed earlier, the cost of copying is a dominant cost in high speed networks. The direct forwarding approach leads to the reduction in number of copies from between 2 (kernel forwarding) and 4 (user program forwarding) to 0 in both cases.

At the end points of communication, the advantages of this approach also become evident. The data that the network protocol software needs to examine is already in the second level cache, which saves expensive cache misses. It may also be possible to re-map the data to where it is required in physical memory without having to copy the data. The advantage of re-mapping depends on the cache line size. The size has to be large enough that the cost of re-mapping is smaller than the cost of a copy.

The cost of reordering and/or assembly and error checking of received packets represents the largest single cost in most protocols. The cost comes from the need to copy data while reassembling or reordering messages and reading every byte to do checksum computation. Dealing with errors represents another area in which protocol design is very important. The exact nature of the underlying network determines the most likely approach for handling packets with errors. Some methods include error correction, time-out to retransmit nega-

tive acknowledgments requesting retransmission, and in some cases, simply discarding the error packet.

The factors that influence the way in which errors are handled are related to the use the data is to be put to, the underlying network, and the importance of the data. Clearly no retransmission can be requested if a network is unidirectional such as any broadcast data stream. If the data is important, error correction information or redundant packets can be sent. For some data types, retransmissions are of little or no value. Situations such as these exist when time sensitive data such as stock prices are being transmitted or in the case of real time data such as audio or positional information. In these cases, the usefulness of the particular packet diminishes as time passes since more up to date information would become available.

The choice between using retransmission based on time-outs or negative acknowledgments requesting retransmission depends on network characteristics and protocol characteristics. An example: if the network is considered to be reliable, then retransmission should be infrequently required and the protocol should not send a retransmission unless it is requested.

There are several costs associated with unnecessary retransmission: the usage of network bandwidth, the usage of CPU and memory resources at both the sending and the receiving machines (in multicast transmissions multiple receivers can be affected.) In a multicast protocol, negative and positive acknowledgments can result in an unnecessary increase in network traffic on unreliable networks. It is important to realize that network errors not only affect network packets that con-

tain data, but also cause acknowledgments and requests for retransmission to get lost.

The VMTP protocol achieves a compromise between these two approaches by requiring that replies to requests must arrive within a time-out interval, or else the request will be retransmitted. Acknowledgment of the reply is implicit in the next request. If a reply is delayed at the server due to work that is required in performing the request, a message is sent to the client on receipt of the retransmission that tells the client that the request was received and is being processed. The client then exponentially backs off the retransmission time-out and the cycle is repeated.

The reason for repeated retransmission is that the reply could have been lost. If the packet is lost, when the retransmission request arrives, the missing parts of the message are selectively resent to the server.

If the reply goes unacknowledged for the time-out interval, the reply is retransmitted, which causes the client to reply with an explicit acknowledgment. The difficulty of setting time-outs is that in some cases this system causes problems.

The use of exponential back off in retransmission causes problems in discovery protocols (protocols designed to find required services), and this is one of the uses for which the VMTP protocol is used. The problem arises in the case that no server exists to respond to the requests. This is a common practice when the desire is to find a unique name.

To find a unique name, a request is sent asking for a reply to the name that has been generated. If no reply is received in the total time-out interval, the name is considered unique. If the time-out is increased exponentially or if the interval is too long, the time required for the

discovery of a unique name is far longer than is desired or practical. If the number of transmissions is too few or not spaced far enough apart, then there is the possibility that the name will not be unique because of temporary problems in the network. This problem is made greater if the network is big or if the requests are sent over multiple networks. Multiple networks produce the worst case since local traffic on some networks could cause congestion and packet loss while other networks are lightly loaded and respond much quicker.

The quality of service provided and the guarantees made by the various levels of the network system influence the amount of work that the protocol has to perform. Reordering of packets is expensive: if the network delivers packets in order, it is one less task required of the receiving end. The reason reception is more expensive than transmission is because the computer that is transmitting data is in complete control of almost every aspect of the communication while sending. The receiver has to be able to handle a variety of data rates, packet ordering, message sizes and errors over which it has little control. The amount of software that is required to handle every conceivable situation that may come up is quite large.

Since it is not possible to have a usable computer network without all the relevant issues being taken care of, it is more of a distribution problem in which the division of labor must be reconsidered to get the most out of high speed networks. The issue comes down to what layer of the network has the required information to best accomplish each of the required functions. Some issues are fairly simple, for example, routing should be done as low as possible in the communications stack, at the first level that has the relevant information to be able to

forward the message to the correct destination. Other issues are more complex.

In the case of error handling, for example, applications may have information that allows them to discard packets or continue operation while waiting for more information. The largest disadvantage of forcing applications to take care of these details is that simple applications do not want to have to take care of all the complexities that are required.

Given the differences in requirements between applications, the simple options seem to be to offer a range of services - perhaps in the form of a variety of linkable libraries that do the necessary work for any given service. These services would make use of a very simple underlying network protocol which provides few or no guarantees about service quality. This approach would prevent the duplication of effort that goes on in many systems at the moment.

Summary of Results

This section is a summary of the most important results obtained during the research for this thesis. The following are the key results for ATM results:

1. The percentage of transmission time consumed by copying data varied from 40% to 80% with quick convergence towards 80%. (See Figure 4, **"Percentage Copy vs. Packet Size,"** on page 62.)
2. The throughput varied slightly as the last ATM cell for each packet filled. Average megabits per second converged to a performance level limit by machine performance. (See Figure 5, **"Megabits per Second vs. Packet Size,"** on page 64.)

3. The cost of packet fragmentation into ATM cells does not increase as packet size increases. This is consistent with point number 2. (See Figure 6, **"Packets per Second vs. Packet Size,"** on page 67.)

4. For very small packets the non-copy overhead dominates the processing cost. For most packet sizes the copy cost dominates the other overhead costs. Both the copy and the non-copy overhead costs increase linearly with packet size. (See Figure 7, **"Time per 100,000 Packets,"** on page 69.)

The remaining results are the key results for VMTP measurements:

1. The total throughput of applications using VMTP increases as the individual packet size increases. In other words, sending a few bigger packets is better than more, smaller packets. (See Figure 8, **"Megabits per Second vs. Packet Size,"** on page 73.)

2. Maximum network utilization is achieved by several VMTP clients communicating simultaneously. This is due to the communication latency of the request-reply protocol. Due to the performance and low latency of VMTP, very high network utilization (65%) is obtained with a single client. (See Figure 9, **"Total Throughput for Multiple Clients,"** on page 76.)

3. Maximum network utilization is achieved at the expense of greater total overhead. This means that as more clients are added, performance for each client decreases and total machine load increases by more than the increase in network performance. (See Figure 10, **"Packets/client/second vs. Number of Clients,"** on page 78.)

4. Total output stabilizes quickly at a constant throughput as more clients are added. This supports number 3 because it shows that the extra expense of adding new clients provides reducing improvements in performance until network saturation is reached. (See Figure 11, **"Packets/Host/Second vs. Number of Clients,"** on page 80.)

5. Latency of request-reply communication increases as packet size is increased. This is due to increases in the transmission time and copy time. (See Figure 12, **"Request-Reply Latency vs. VMTP Packet Size,"** on page 82.)

Conclusion

This thesis has presented a discussion of the issues involved in network protocol design. Measurements have also been made of various aspects of network and protocol performance. Finally, papers and texts which deal with similar and related issues have been examined to get a broader picture of these issues and also to back up the results of the investigation.

It is clear that the design of a network protocol is not a simple task. There are a great variety of situations, desirable features and constraints that exist in any real world application. Together these factors make it impossible to design a single protocol that provides all the desired functionality for every use and application.

The performance of the networks that exist easily spans a range of 7 orders of magnitude. It is not possible to present a protocol design that can be used across the entire range of networks. We have shown some of the problems associated with such a range of performance. At the

ends of the communication speed range, the constraints on the system are totally different. Very few machines are able to handle data in the gigabit per second range and few can supply data at this sort of rate. On the other end of the speed range, many applications cannot operate in any reasonable way over a 300 bits per second link.

Network latency is another variable that exhibits great variability. Most networks operate with sub-second latencies, although latencies in the low seconds are found in many wide area networks when loads are high. Networks guaranteeing delivery of messages can exhibit latencies of hours or even days when partial network failures occur. This range may be acceptable to a mail program but completely unacceptable to an interactive program. The mail application would not want notification of network failure if a message was going to be delayed for an hour, but an interactive application would want notification of communication failure if it was to face the same 1 hour delay.

A related and often closely coupled issue is the configuration of the network. Local area networks usually differ in almost all respects to wide area networks. In distributed systems that span a wide area, the communication model at the application level has to be suited to both the wide area and the local area network characteristics. These details influence the design of the underlying protocols.

The above are constraints placed on the protocol by the physical network construction and design. Further constraints are imposed on the protocol by the needs and demands of the applications that are to be written on top of them.

The delivery of data from a sender to a receiver cannot be considered the basic purpose of a network protocol since different applications

have different definitions of what the same simple phrase actually means. Two distinct ways of communicating exist: publish-subscribe and request-reply. The uses for each are largely disjoint from the uses for the other, but considerable overlap exists: a class of needs exists that can be fulfilled by either paradigm. In fact, almost all problems that can be solved using one method can be solved using the other, but here we start to see that the inappropriate choice in the design can be costly in terms of performance and wasted resources.

The use of a point-to-point request-reply application to disseminate information to a large number of applications would require a large number of connections and a large number of copies of the data to be sent across the network. This solution clearly does not scale well given any limit to the available bandwidth in the network. Similarly, the use of broadcast to communicate between two applications potentially fills the entire network, including all subnets, with messages that are of no interest to the majority of machines. This consumes network bandwidth and machine resources throughout the network. The design of intermediate solutions that offer different blends of the extremes of functionality yield possible solutions. A few simple communication models are one-to-one, one-to-many and one-to-(one-of-many).

The next set of constraints that are imposed on the protocol are those of application performance desires. The speed of delivery, the guarantee of delivery, handling different priority communications and/or confirmation of message delivery. Each desire places additional constraints and complexity on the protocol design and implementation essentially requiring very efficient use to be made of the limited resources that are available in a network system.

Handling various failures and errors that occur in the system is another constraint requiring more design effort to create guarantees and timely delivery in complex systems. The design of the protocol should include an understanding of possible errors and methods of handling the errors in ways that give applications the most freedom to deal correctly with them. As we have seen earlier in the thesis, these constraints require the use of valuable network resources, and, as such, constrain the network performance. The exact design of systems to handle problems is thus another constraint on the systems performance.

The final set of design issues are the requirements for network security. Security includes access control to data on the network and encryption of data to prevent access to data. Security requirements complicate protocol design and reduce network performance through the added steps and data transformations that are required.

Given all the above constraints, it is clear that the design of the protocol from an application standpoint is complex, and we have argued that most of the decisions not only impact the ease of use and utility of the protocol and thus the value of the communications infrastructure. The other side of the coin is the consideration of the interaction between protocol design and network performance.

We have argued that at the lowest levels the performance that can be achieved in all network systems is strongly influenced by the protocol design. The discussion of the motivation for various design decisions in regards to the application is to give a context and functional constraint that is placed on the protocol design. This removes from discussion the possibility of custom high performance solutions that

work for a single particular application, but do poorly when used for other purposes. Protocols designed in that manner are designed for specific narrow requirements or research needs and have very little use in real networks that are expected to handle a wide variety of traffic.

The examination of ATM network drivers illustrated a network in which the machine attached to the network was unable to utilize the full available bandwidth of the network. We demonstrated that despite all our efforts, and the theoretical discussions of the machines potential, it was not possible to drive the network at the limit. We also argued that this limitation was largely created by the way the protocol design and hardware implementation of the equipment constrained copy performance. Alternate designs were discussed that could lead to better utilization of the network.

The examination of VMTP design and the discussion of the work of expanding the functionality of the protocol illustrated the costs of providing a large number of features and the limitations that are imposed by the operating system which are made worse by these design decisions. The VMTP protocol is outwardly simple and elegant, but as features are added, the implementation grows more complex. Eventually, kernel constraints started to play a role in the performance of the protocol. These constraints could largely have been removed if the design and implementation of the protocol were changed.

The success of TCP/IP is testimony to its relative simplicity and adaptability. Features have slowly been added to IP over the years to meet the new and changing needs of the network community. Part of its success was due to the nature of the network and the applications

that used it. Network bandwidth was free to many end-users and thus a wide range of performance was acceptable. The network and the protocol effectively grew together, each influencing the other. With the growth of the Internet, expectations also grew. The lack of preconceived performance goals allowed easier growth in the protocol. The current growth in commercial interest in the Internet is driving the need for increased performance.

In the future, it is likely that performance will continue to be important, while bandwidth will become cheaper. Demands on networks will be higher and thus the performance and network efficiency of protocols will become ever more important. The requirements of the machines that are connected to the network will also become important. As networking becomes more widespread, the motivation to connect smaller and less powerful computers will grow. It is conceivable that this trend will include 'smart appliances' at some point. These less powerful machines will rely on a protocol that does not place undue demands on their limited resources.

We have shown that protocol design is a compromise. We have also shown that the protocol influences the performance of the network. The forces driving for more features in the protocol detract from the overall system performance. We have shown that this occurs because of the costs associated with handling the protocol. Protocol design, network design, network performance and application performance are all interlinked. The importance of communication is increasing, and therefore the importance of protocol design to maximize communication performance is also growing. Understanding the influence of protocol choice on network performance is growing in importance.

This thesis is meant to help with an understanding of some of the issues involved.

Reference

[1] Alan Berenbaum, Joe Dixon, Anand Iyengar, and Srinivasan Keshav.
A Flexible ATM-Host Interface for XUNET II.
IEEE Network 7(4) July 1993.

[2] Eric J. Berglund.
An Introduction to the V-System.
Distributed Systems Group, Stanford University.

[3] Dimitri P. Bertsekas and John N. Tsitsiklis.
Parallel and Distributed Computation - Numerical Methods.
Prentice Hall, 1989.

[4] David R Cheriton.
Serpent: A High-Performance Internetworking Approach.
Technical Report STAN-CS-89-1273, Computer Science Department, Stanford University, 1989.

[5] David R. Cheriton.
The Unified Management of Memory in the V Distributed System.
Technical Report STAN-CS-88-1192.
Computer Science Department, Stanford University, 1988.

[6] David R. Cheriton.
The V Distributed System.
Communications of the ACM, 31(3), pages 314-333, March 1988.

[7] David R. Cheriton.
VMTP: Versatile Message Transaction Protocol.
Protocol Specification, Distributed Systems Group.
Stanford University, 1988.

[8] David R Cheriton, H. A. Goosen, and P.D. Boyle.
Paradigm: a highly scalable shared memory multicomputer architecture.
IEEE Computer 24(2), pages 33-46, February 1991.

[9] David R. Cheriton and Dale Skeen.
Understanding the Limitations of Causal and Total Ordered Communication.
Proceedings of the 14th ACM Symposium on Operating Systems Principles, pages 44-57, December 1993.

[10]David R. Cheriton and Carey L. Williamson.
Network Measurement of the VMTP Request-Response Protocol in the V Distributed System.
AMC Sigmetrics, 1987.

[11]David R. Cheriton and Carey L. Williamson.
VMTP as the Transport Layer for High-Performance Distributed Systems.
IEEE Communications, June 1989.

[12]Douglas E. Comer and David L. Stevens.
Internetworking with TCP/IP - Volume III: Client-Server Programming and applications.
BSD socket Version.
Prentice Hall, 1993.

[13]Chris Dalton, G. Watson, D. Banks, C. Calamvokis, A. Edwards, and J Lumley.
Afterburner.
IEEE Network 7(4) July 1993.

[14]Peter Druschel, Mark B. Abbott, Michael A. Pagels and Larry L.
Peterson.
Network Subsystem Design.
IEEE Network, 7(4) July 1993.

[15] Zygmund Haas and David R. Cheriton.
Blazenet: A Packet-Switched Wide-Area Network with Photonic Data Path.
IEEE Transactions on Communication, 38(6). June 1990.

[16]Gerald W. Neufield, M. R. Ito, M. Goldberg, M. J. McCrutcheon, and S Ritchie.
Parallel Host Interface for an ATM Network.
IEEE Network 7(4) July 1993.

[17]J. K. Ousterhout.
Why Aren't Operating Systems Getting Faster as Fast as Hardware?
USENIX Conference Proceedings - June 1990.

[18]Craig Partridge.
Fixing the Internet Protocol.
IEEE Network 7(4) July 1993.

[19]Craig Partridge.
Gigabit Networking.
Addison-Wesley Publishing Company, 1994.

[20]S. Przybylski, M. Horowitz, and J.L. Hennessy.
Performance tradeoffs in cache design.
In Proceedings 15th International Symposium on Computer Architecture, pages 290-298, May 1988.

[21]Dale Skeen.
An Information Bus Architecture for Large-Scale, Decision-Support Environments.
USENIX Conference Proceedings, Winter 1992.

[22]Jonathan M. Smith and Brendan S. Traw.
Giving Applications Access to Gb/s Networking.
IEEE Network 7(4) July 1993.

[23]William Stallings.
Data and Computer Communications.
MacMillan Publishing Company, 1991.

[24]W. Richard Stevens.
UNIX Network Programming.
Prentice Hall, 1990.

[25]Andrew S. Tanenbaum.
Computer Networks.
Second Edition.
Prentice Hall, 1988.