# Performance Analysis of a Software Implementation of the Xpress Transfer Protocol

---

A Thesis

Presented to

the Faculty of the School Engineering and Applied Science

University of Virginia

---

In Partial Fulfillment

of the Requirements for the Degree

Master of Science (Computer Science)

by

Timothy W. Hartrick

August 1991

# APPROVAL SHEET

This thesis is submitted in partial fulfillment of the

requirements of the degree of

Master of Science (Computer Science)

_____

Timothy W. Hartrick

This thesis has been read and approved by the Examining Committee:

_____

Thesis Advisor

_____

Committee Chairman

_____

Accepted for the School of Engineering and Applied Science:

_____

Dean, School of Enginneering and Applied Science

August 1991

## Abstract

In this thesis we present a survey of a group of influential Transport Layer protocols, including the Transmission Control Protocol (TCP), ISO Transport Protocol Class 4 (TP4), the Versatile Message Transaction Protocol (VMTP) and the Xpress Transfer Protocol (XTP). A variety of features of each protocol are examined in detail. These features include packet formats, communication syntax, error and sequencing control and multicast capabilities. Next, we describe an implementation of XTP version 3.5 which was developed at the University of Virginia Computer Networks Laboratory. We discuss the results of a set of performance experiments performed on UVa XTP 3.5. These experiments include unicast throughput and latency measurements, and multicast throughput measurements. We also present a set of performance measurements which compare the performance of a commercially available implementation of TCP with the performance of UVa XTP 3.5 running on the identical hardware platform.

## Acknowledgments

I would like to express my gratitude to my advisor, Dr. Alfred C. Weaver, for his generosity and encouragement throughout my tenure in the Computer Networks Laboratory. I would also like to thank my colleagues in the Computer Networks Laboratory, particularly Robert Simoncic and John Fenton for developing the UVa XTP 3.5 implementation. Their insight into the internal workings of the implementation was invaluable. Special appreciation goes to Molly and Bert Dempsey for providing me with a home during the completion of this work. Finally, I would like to thank my parents for their support and encouragement.

# Table of Contents

# List of Figures

# List of Tables

*Chapter 2*

*Chapter 1*

# Transport Protocol Survey

## 1.1. OSI Reference Model

In the late 1970's the International Standards Organization (ISO) began work on a set of protocol standards for computer and data communications. This effort was undertaken as a response to the need for a set of data communications standards which were neither proprietary to a particular vendor, nor of interest only to a small subset of the international community. The architectural structure for the development of this set of communication standards is the seven layer Open Systems Interconnection Architecture Basic Reference Model [13]. Figure 1.1 contains a graphical depiction of the OSI reference model protocol stack.

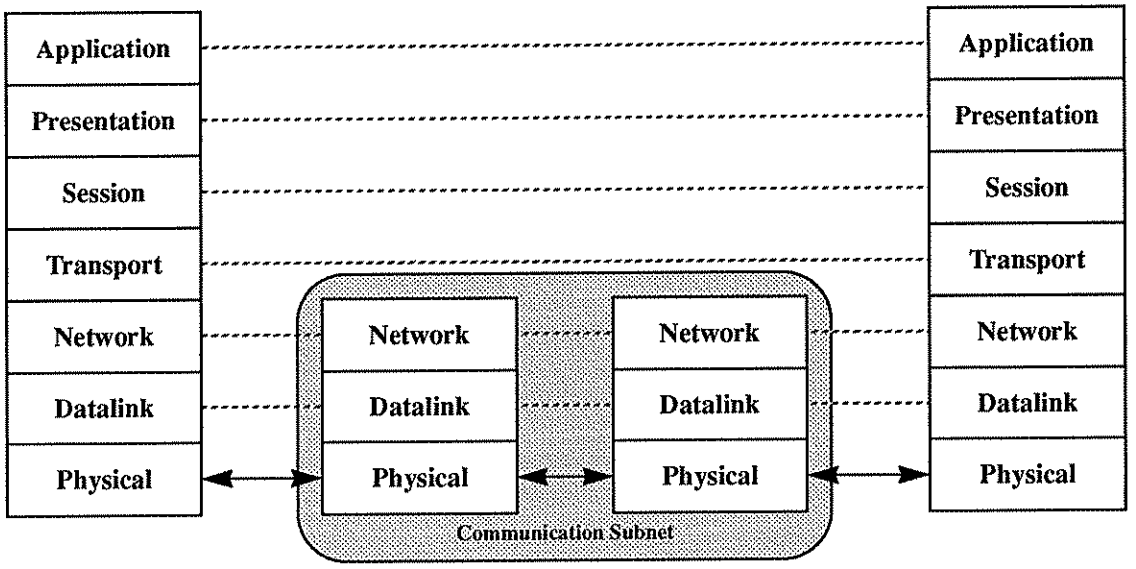


**Figure 1.1**— *OSI Reference Model*

The key feature of the OSI architecture is the strict division between each of the seven layers of service. Each layer in the OSI protocol stack provides a set of services to the layer above and requests services from the layer below. The interface to each layer is a set of primitives which are part of the protocol service specification. Information about the state of each layer is isolated from the layers above and below with the exception of what state information can be inferred by the use of the defined primitives [1, 18, 21, 28]. The following subsections contain a brief discussion of the services provided by each of the seven layers of the OSI reference model.

### 1.1.1. Physical Layer

The physical layer of the OSI reference model provides a set of services which allow for the transmission of a serial bit stream across some physical media.The physical layer is principally concerned with the electrical and mechanical characteristics of the signalling media.

### 1.1.2. Datalink Layer

The datalink layer is concerned with utilizing the serial bit stream services provided by the physical layer to provide data communication services along a single network link. These services are generally provided by taking the raw bit stream capability of the physical layer and combining sequences of bits into datalink protocol data units or frames. These frames can contain both datalink control information and user data.

In the case of local area networks the datalink layer is divided into two sublayers known as logical link control (LLC) and medium access control (MAC). Logical link control provides reliability and acknowledgments while medium access control provides an unreliable packet communications facility over a multidrop physical link. The most common local area network standards are the IEEE 802.x local area network protocols.

These protocols have been adopted by ISO for use as datalink layers [22, 23, 24, 25]. Figure 1.2 contains a graphical depiction of how the IEEE 802.x standards fit into the ISO architecture [25].

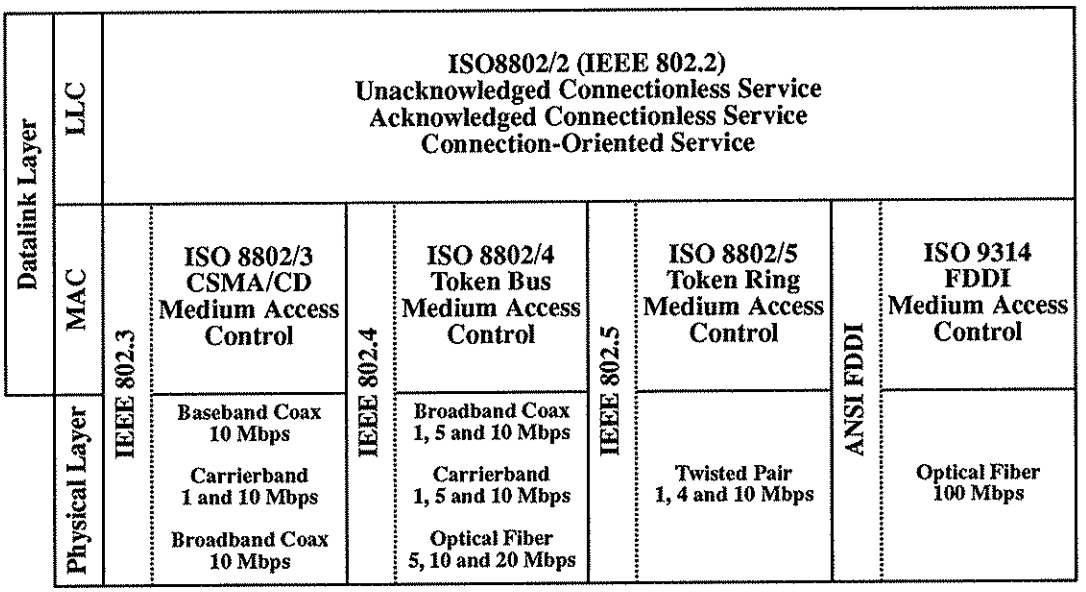


**Figure 1.2** — *OSI Datalink Sublayers*

### 1.1.3. Network Layer

The network layer provides addressing and routing services which allow stations, not physically connected to the same network link, to communicate with each other. It provides the transport layer with a variety of connectionless and connection-oriented services including an unreliable data delivery service. By unreliable we mean that network a protocol data unit (NPDU) may be lost, reordered with other NPDUs, or duplicated while in transit to the destination.

### 1.1.4. Transport Layer

The transport layer provides full end-to-end communication between stations using a network layer data delivery service. This layer is extremely important because it is charged with providing end-to-end reliable data transmission services over a potentially unreliable network service and at the same time conceals the nature of the underlying network. The transport layer must also provide some end-to-end flow control mechanism so that fast transmitters can not overrun the buffer capacity of slower receivers.

### 1.1.5. Session Layer

The session layer provides various connection management services. In particular it is responsible for providing either full-duplex or half-duplex transport connections and also manages the graceful closing of transport connections.

### 1.1.6. Presentation Layer

The presentation layer is responsible for providing the application layer with a set of datatype translation services which allow computer systems with incompatible datatype representations to communicate with each other via the ISO protocol stack. This set of services allows heterogeneous computer architectures and operating systems to transmit complex data structures between systems and have the content and meaning of the data structure preserved.

### 1.1.7. Application Layer

The application layer provides a predefined set of services which can be accessed by an application programmer. Examples of these types of services are file transfer protocols and electronic mail services.

## 1.2. Transport Protocols

Transport layer protocols have been the focus for a substantial amount of research over the last twenty years. Because they provide peer-to-peer communications while keeping the nature of the underlying network transparent to the layers above, they present unique challenges to the protocol designer.

In the sections which follow we will examine the design and function of a set of influential transport protocols. These influential protocols will range from traditional connection-oriented protocols, to special purpose transport protocols used for distributed systems and embedded real-time systems. We will conclude with a description of the Xpress Transfer Protocol, a high-speed transfer layer protocol which employs many of the design concepts from the set of influential transport protocols.

### 1.2.1. Transmission Control Protocol

During the mid-1970s the Defense Advanced Project Research Agency (DARPA) began development of a set of data communication standards for use within the military. This development was driven by the need to support communication and inter-operability among the multi-vendor computer systems in use by the military. By 1978-79 the development efforts had spawned a set of five military standards. The standard of interest in this section is MIL-STD-1778, the Transmission Control Protocol (TCP). In the OSI reference model TCP is a transport protocol, although it has not been adopted as an ISO standard. The unreliable network data delivery service associated with TCP is defined in MIL-STD-1777 and is known as the Internet Protocol (IP). Because TCP and IP provide a large portion of the functionality of the military communications standards, the whole suite is usually referred to as TCP/IP [8, 15, 23].

TCP is a connection-oriented sliding window protocol which uses sequence numbers, positive acknowledgments and timer based retransmission to guarantee reliable service. It also implements a buffer reservation flow control mechanism in order to minimize the loss of data between stations which are ill matched with respect to performance. Each TCP connection provides full-duplex octet stream communication between the two endpoints of the connection. In the sections which follow we will discuss some of the details of the protocol with emphasis on segment formats, connection establishment and termination, and the mechanisms used to provide reliable service.

#### 1.2.1.1. Segments and Segment Structure

TCP uses the unreliable datagram service provided by IP. Each IP datagram has a header portion and a data portion. TCP uses the data portion of the IP datagram to hold its own header information and optional user data. The combination of a TCP header and optional data is called a TCP segment. In Figure 1.3 we see the format of a TCP segment header.

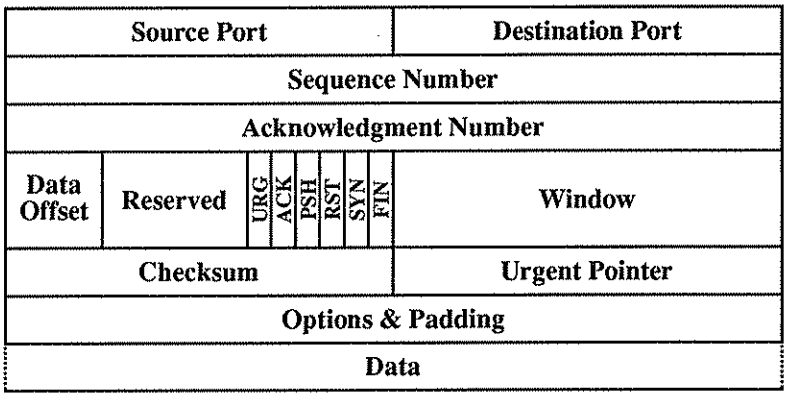


**Figure 1.3**—*TCP Segment Header*

- The 16-bit source port field specifies the port on the source host from which the segment originated. The destination port field specifies the port on the destination host for which the segment is intended. Host addressing information is not required in the TCP segment header because it is contained in the IP datagram header which encapsulates TCP segments.
- The sequence number field is a 32-bit number which specifies the sequence number of first octet of data in this segment. TCP uses octet significant sequence numbers to keep track of the current position in the data stream.
- The acknowledgment number is the sequence number of the next octet of data expected by the source port end of the connection. TCP provides for the piggybacking of acknowledgments on outgoing data segments in order to provide more efficient full-duplex operation.
- The 4-bit data offset field contains a count of the number of 32-bit words contained in the header. This field is required because the options field can have variable length.
- The 6-bits after the reserved field are the flags. The flags are used to indicate which fields of the header are significant in this segment and also allow protocol control information to be related to the destination port end of the connection.
- The urgent flag (URG) indicates that the urgent pointer is significant in this segment. The urgent pointer itself is discussed later.
- The acknowledgment flag (ACK) is used to indicate that the acknowledgment number field is significant in this segment. This flag indicates to the receiver of the segment whether or not the segment is being used to piggyback acknowledgments.
- The push flag (PSH) is used to indicate that this segment should be delivered in an expedited fashion. It provides a way to avoid buffering latency but still delivers the data in sequence.
- The reset flag (RST) is used to request from the destination port end of the connection an abnormal connection closure. The reset is issued when the source port end of the connection has experienced system failure and recovered quickly. Since the source port end of the connection cannot guarantee its state after the failure, it responds to all incoming segments by transmitting a segment with the reset flag set. This indicates to the destination port end of the connection that the connection has failed and should be closed abnormally.
- The synchronize flag (SYN) is used to request from the receiver a synchronization of sequence numbers. This flag is used during connection establishment.
- The finished flag (FIN) is used to indicate to the receiver that no more data is available. The flag is used to gracefully close one half of the full-duplex communication channel.
- The window field is used to implement end-to-end flow control. The 16-bit number indicates the number of data octets for which the destination port end of the connection has buffer space.
- The 16-bit checksum field is calculated over the TCP segment padded to the nearest 16-bit word with a 12-octet pseudo-header appended to the beginning of the segment. The pseudo-header contains IP addresses and other information similar to that contained in a IP datagram header. Neither the pseudo-header or the padding is actually transmitted with the segment. They are reconstructed on the destination port end of the connection.

- The urgent pointer field is a 16-bit number which points to the first octet of data which follows a sequence of urgent data octets. Data marked as urgent receives priority transmission and delivery service. It is different from a segment with the push flag set because urgent data receives priority service over normal data.
- The options field is a variable length field which is padded out to the nearest 16-bit word for checksum calculation purposes.

#### 1.2.1.2. Connection Establishment

The endpoint of a TCP connection can be opened by the user in one of two states. The connection endpoint can be opened in a passive state in which it is listening for a connection request from a remote connection endpoint or it can be opened in an active state in which it is sending a connection request to a specified connection endpoint. A connection can be established between two connection endpoints in one of two ways, but in either case connection establishment involves at least a three-way handshake between the two endpoints before the connection is fully established. Figure 1.4 shows two possible sequences of events which could lead to successful connection establishment.

In Figure 1.4a, A transmits a SYN segment and *i*, which is the initial sequence number value for the A-to-B half of the full-duplex connection. When B receives this segment it responds with a segment with the SYN and ACK flags set and *j* in its sequence number field. The value *j* is the initial sequence number value for the B-to-A half of the full-duplex connection. In addition, the acknowledgment number field contains *i*+1, in order to acknowledge the initial value of the A-to-B sequence number. When A receives the acknowledgment from B, it responds with an ACK segment with *j*+1 in its acknowledgment number field in order to acknowledge the initial sequence number value for the B-to-A half of the connection. This segment may also contain data and a sequence number field of *i*+1.

In Figure 1.4b, A and B both send SYN segments simultaneously with initial sequence number values of *i* and *j* respectively. These initial sequence number values are

acknowledged with ACK segments with the appropriate values in the acknowledgment number fields. Once the acknowledgments are received on both sides the connection is established and data may be transferred using the agreed initial sequence number values.

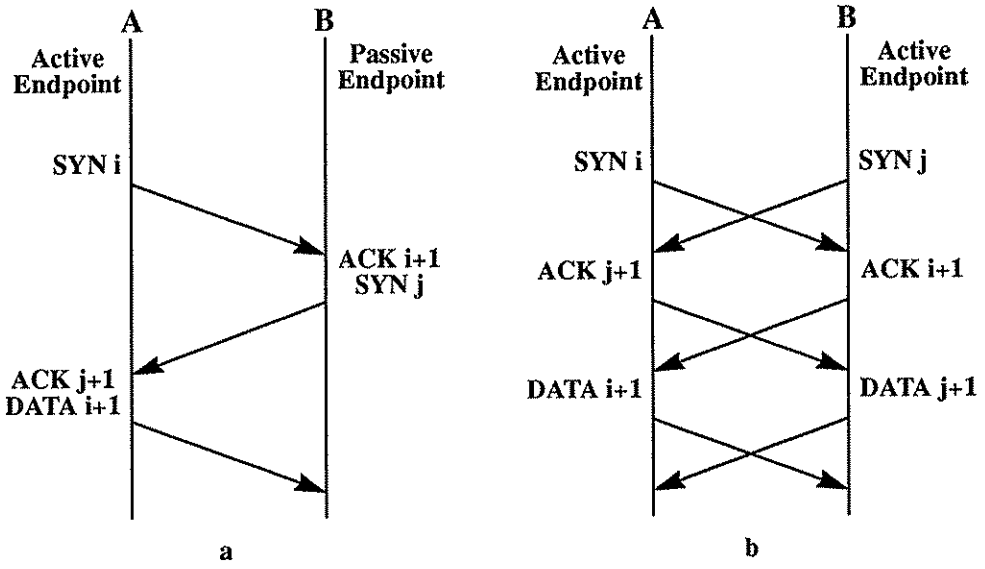


**Figure 1.4** — *Connection Establishment Syntaxes*

### 1.2.1.3. Connection Termination

Connection termination in TCP is very similar to connection establishment in that it requires at least a three-way handshake between the connection endpoints. Figure 1.5 shows one possible sequence of events which would lead to a graceful connection termination.

In order for a connection to be closed, each endpoint must transmit a FIN segment with the current sequence number and receive an ACK segment with the current sequence

number in the acknowledgment number field. In this way a graceful termination of the connection with no loss of data is guaranteed.

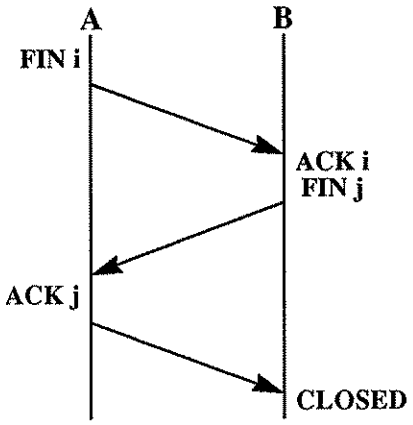


**Figure 1.5** — *Connection Termination*

### 1.2.1.4. Windows, Acknowledgments and Retransmission

When using an unreliable network datagram facility like IP, TCP segments can be lost or delayed for an indeterminate amount of time. To combat this TCP employs a system of sliding windows, acknowledgments and segment retransmissions in order to guarantee reliable service.

A sliding window is a portion of the sequence space which TCP transmits as a segment or group of segments. Figure 1.6 shows both the transmitter and the receiver windows moving through the sequence space. At the transmitter side each window has associated with it a retransmission timer. When the segments in the window are transmitted the timer is started. If an acknowledgment is received, the portion of the segments which have been acknowledged are removed from the window. Any data which has become available at the front edge of the window is transmitted and the timer is restarted. If the

timer expires then all of the unacknowledged data segments in the window are retransmitted. This is known as the go-back-n retransmission strategy.

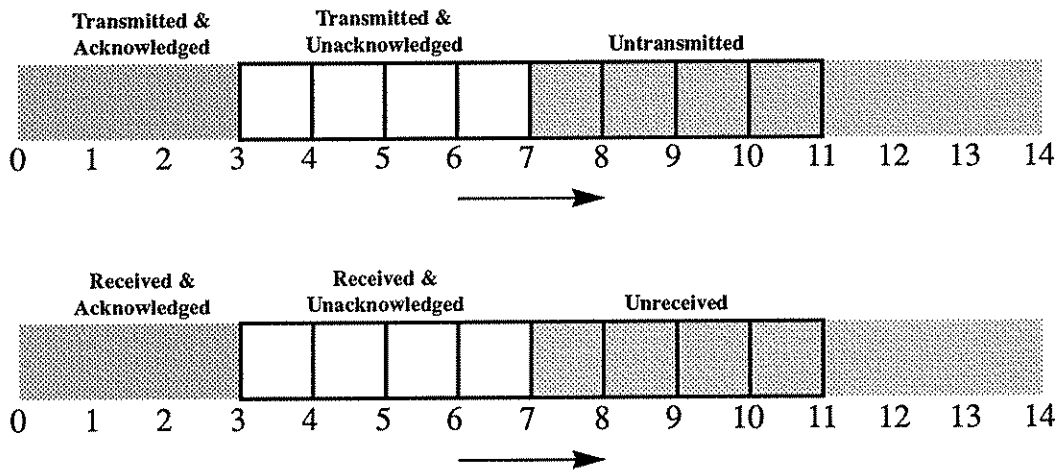


**Figure 1.6** — *Transmit and Receive Windows*

The acknowledgment strategy used by TCP is fairly simple and involves no timers When a full window of data is received the receiver responds with an acknowledgment. Acknowledgments can be piggybacked with outbound data segments in order to improve network efficiency.

### 1.2.1.5. Flow Control

Because different computer systems have very different protocol processing capabilities with respect to the number of segments which can be processed and the amount of data that can be buffered, TCP implements an end-to-end flow control mechanism based on a window credit system.

Each end of a TCP connection has a window allocation for receiving data. When a window of data is acknowledged, the acknowledgment segment contains in its

acknowledgment number field a sequence number corresponding to the next octet of data expected by the receiver. In the window field of the acknowledgment segment is a value corresponding to the number of octets after the acknowledgment number that the receiver can accept in its window. For example, if the acknowledgment number field contained the value *i* and the window field contained the value *j* then the transmitter can transmit data segments containing sequence numbers between *i* and *i+j*-1 and nothing more until another acknowledgment is received with further credit. In this way a transmitter can be throttled in order to keep the receiver from being forced to drop error-free segments because there is no available buffer space.

### 1.2.2. ISO OSI Transport Protocol Class 4

The OSI architecture has defined five classes of connection-oriented transport protocol service numbered 0 through 4 [14]. The differences between these five service classes are related to the type and quality of service required by session layer, and more directly, by the quality of service provided by the network layer. ISO transport protocol class 4 (ISO TP 4) assumes that the network layer provides nothing more than an unreliable datagram delivery service. The network service usually associated with TP4 is the ISO connectionless network protocol (CLNP) which provides services very similar to IP.

ISO TP4 is very similar to TCP in many ways. However, there are some rather important differences which should be noted at the outset. First, ISO TP4 is not a octet stream based protocol; it is based on transport protocol data units (TPDUs). This means that ISO TP4 sequence numbers are not assigned on a per-octet basis but rather on a per -TPDU basis. Secondly, because ISO TP4 is one of a whole class of transport protocols which all use the same set of TPDU formats, the TPDU formats are much more varied and contain more options and optional fields [1, 14, 21]. In our discussion of the TPDU formats and

general protocol mechanisms, we will attempt to concentrate exclusively on the issues which concern ISO TP4.

### 1.2.2.1. Transport Protocol Data Units

ISO TP4 supports nine different TPDU types. Each type has a different header format. Like TCP, the use of options in the header formats allows some headers to have variable length. Figure 1.7 shows the fixed length portions of the TPDU header formats used by TP4. We will briefly describe the function of each of the fields in the packet header formats.

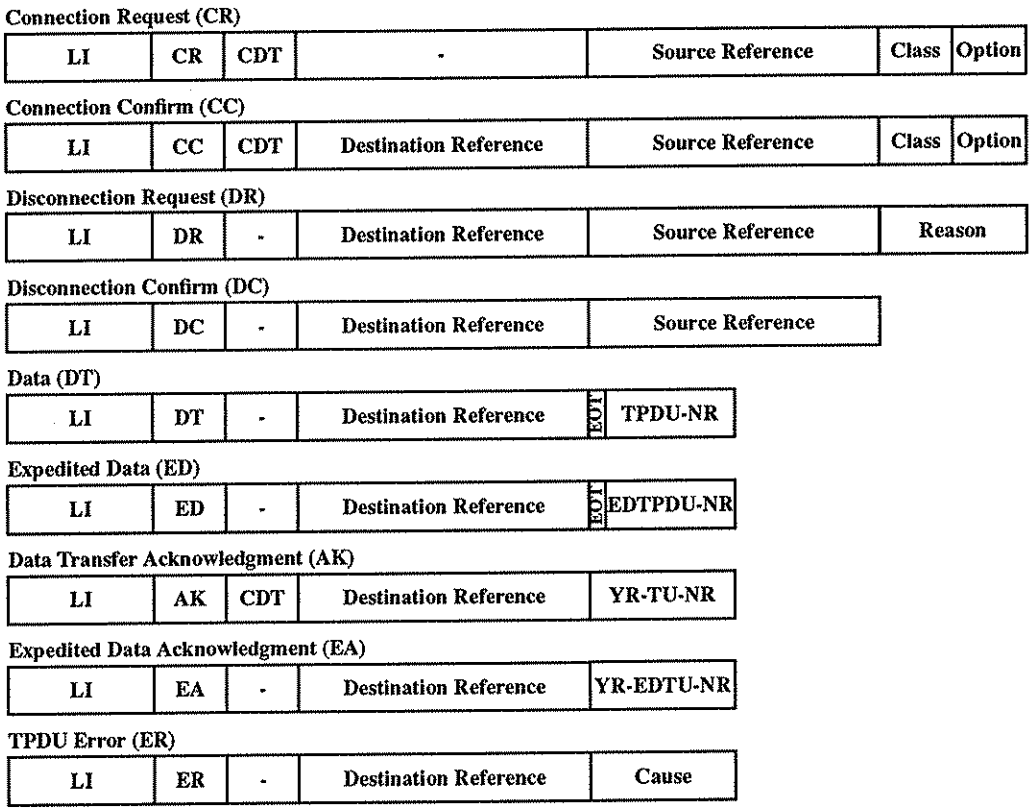


**Figure 1.7** — *TP4 TPDU Headers*

- Each TPDU begins with an 8-bit length indicator field (LI). This field contains the length of the TPDU header, including the length of any options but not including the length of the LI field.
- The 4-bit TPDU type field is used to indicate which type of TPDU (e.g., CR, CC, DR, etc.) has been received.
- The 4-bit credit field (CDT) is used to establish the flow control buffer credit for each end of the connection.
- The 16-bit destination and source reference fields are used by the transport entities on either end of the connection to distinguish the connection with which the TPDU is associated.
- The 4-bit class field indicates which class of service the connection is using.
- The 4-bit options field indicates which options will be utilized by this connection.
- The 8-bit reason field is used to indicate the reason for a disconnection request.
- The 1-bit end of transmission (EOT) field is used to indicate that the end of a multi-TPDU message has been reached. It is set to 1 on the last TPDU of the group.
- The 7-bit TPDU-NR field is the sequence number of the current data transfer TPDU.
- The 7-bit EDTPDU-NR contains the sequence number of the current expedited data transfer TPDU.
- The 8-bit YR-TU-NR field contains the sequence number of the next expected data transfer TPDU.
- The 8-bit YR-EDTU-NR field contains the sequence number of the next expected expedited data transfer TPDU.
- The 8-bit cause field is used to indicate the reason that a TPDU was rejected.

#### 1.2.2.2. Connection Establishment

The mechanism for connection establishment in ISO TP4 is very similar to that of TCP. The two protocol state machines for connection establishment are virtually identical. The only difference is that TP4 automatically uses zero as the initial sequence value and TP4 uses the connection establishment process to initialize buffer credit (CDT) values used in TP4's end-to-end flow control mechanism.

Connections are established by the initiating transport entity transmitting a CR-TPDU to the destination transport entity. The initiator's CR-TPDU contains an initial CDT value and also contains any options which the initiator desires for this connection. On receipt of the CR-TPDU the destination transport entity assigns the connection a source

reference and responds to the CR-TPDU with a CC-TPDU. The destination's CC-TPDU contains the destination reference extracted from the source reference of the CR-TPDU, and the assigned source reference. It also contains the destination's initial CDT value. When the initiator receives the CC-TPDU in order to complete the three-way handshake and establish the connection, the initiator must respond with a DT-TPDU, ED-TPDU, or AK-TPDU.

#### 1.2.2.3. Connection Termination

ISO TP4 connection termination is quite a bit different than that of TCP. ISO TP4 uses a two-way handshake in order to release a connection. When a transport entity wishes to terminate a connection it transmits a DR-TPDU. Any DT-TPDUs which were queued for transmission previous to the disconnect request are discarded. The receiving transport entity responds to the DR-TPDU by discarding any pending DT-TPDUs and transmitting a DC-TPDU. ISO TP4 does not possess a graceful close mechanism like TCP. Within the OSI reference model the responsibility for graceful closure without loss of data resides at the session layer.

#### 1.2.2.4. Acknowledgments and Retransmission

ISO TP4 uses mechanisms very similar to TCP; however there are some differences. First, TP4 requires that AK and EA TPDUs be transmitted explicitly. There is no piggybacking of acknowledgments on outbound DT or ED TPDUs. Secondly, when the TP4 retransmission timer expires only the first TPDU in the window is retransmitted. TP4 can implement this strategy because the receiver maintains an acknowledgment timer which guarantees the transmission of an acknowledgment for any successfully received data within a certain time interval. If the transmitter's retransmission timer expires then the

transmitter is assured that at least the first TPDU in the window was lost and therefore can retransmit only that TPDU in order to force acknowledgment from the receiver.

#### 1.2.2.5. Flow Control

ISO TP4 flow control is identical to the flow control used in TCP with the exception of the different sequence number systems employed by the two protocols. When an AK-TPDU is received by a transport entity it examines the YR-TU-NR and the CDT of the TPDU. The YR-TU-NR field contains the sequence of the next expected DT-TPDU. The CDT field indicates the number of buffer openings available in the receiver's window. The transmitter is given the right to transmit DT-TPDUs with sequence numbers between YR-TU-NR and YR-TU-NR + CDT - 1, inclusive.

### 1.2.3. The Versatile Message Transaction Protocol

The Versatile Message Transaction Protocol (VMTP) was developed by Dr. David Cheriton for use with the V distributed system at Stanford University [2]. It was designed specifically for use in distributed systems applications using remote procedure calls (RPC) and uses request/response communication semantics rather than the traditional connection-oriented semantics of TCP and ISO TP4.

There were three design goals for the VMTP protocol, all targeted at overcoming the inadequacies of connection-oriented protocols for distributed system applications. First, connection-oriented protocols do not support request/response semantics well and their flow control mechanisms are not sufficient to combat packet overrun problems at the host network interface. Specifically, with high speed, low error rate local area networks the main cause of lost packets is not bit errors on the media, but rather the dropping of packets at the host interface because packets cannot be buffered quickly enough. Second, the current set of protocols do not provide host independent naming facilities which would allow process

migration in a distributed system environment. Lastly, functions such as real-time datagrams, multicast and security are not supported at all by traditional connection-oriented transport protocols and can be very useful in a distributed environment based on remote servers [3, 4, 5]. In the sections which follow we discuss some of the major features of VMTP.

#### 1.2.3.1. Packet Formats

VMTP uses only two packet types—request and response. Figure 1.8 shows the VMTP packet structure. Each VMTP packet is a member of a packet group consisting of one or more VMTP packets. There can be up to 32 packets in a packet group and up to 16K octets of user data. The user segment data in a packet group is divided into a maximum of thirty-two 512 octet blocks. VMTP packet groups can be grouped to form message transactions. Message transactions can consist of as many as 256 packet groups for a total of 4 million octets of data per message transaction. Each VMTP packet contains a header, message control block (MCB), and data segment. The MCB is contained in all packets of the message. We will briefly discuss the functions of the fields which comprise the VMTP packet header, MCB, and data segment.

- The 64-bit client entity identifier is used to identify the client process involved in the message transaction.
- The 3-bit version field identifies the current version of the protocol in use.
- The 13-bit domain field is used to identify the administrative domain responsible for the client and server involved in this transaction.
- The 1-bit HCO flag is used to indicate whether the checksum should be calculated over the whole packet or just the packet header.
- The 1-bit EPG flag indicates whether the packet group of which the packet is a member is using encryption.
- The 1-bit MPG flag indicates whether the packet group of which the packet is a member is a multicast packet group.
- The 13-bit length field indicates the number of 32-bit words contained in the segment data field. All VMTP packets all required to be aligned on an 8-octet boundary.

- The 9-bit flags field under normal circumstances contains all zeros. These flags are used to implement VMTP's transaction streaming capability, request explicit request acknowledgments and other non-standard services.
- The 3-bit retransmission count field is used to indicate the number of times a packet group has been retransmitted.
- The 4-bit forward count field is used to indicate the number of times a request has been forwarded.
- The 8-bit interpacket gap field is used by the transmitter to space packet transmissions so that the receiver can keep up with the flow of data. Each tick represents 1/32 of a maximum sized network packet transmission time. This translates to a maximum interpacket gap time of 8 network packet transmissions.
- The 8-bit PGcount field contains the number of packet groups which are being cumulatively acknowledged by this transaction packet group. This field is used when using streamed transactions.

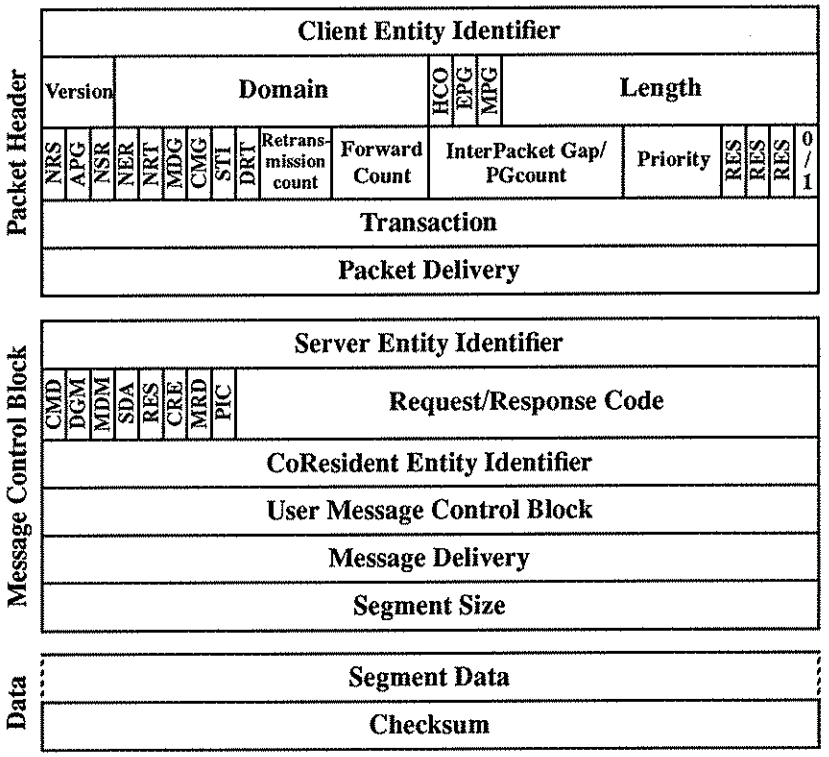


**Figure 1.8** — *VMTP Packet Format*

- The 4-bit priority field is used to indicate the transmission and reception priority level of a request.

- The 1-bit function code field indicates whether the packet is part of a request or response packet group.
- The 32-bit transaction field contains the unique identifier for transactions to which the packet belongs.
- The 32-bit packet delivery is a mask used to indicate which of the 512-octet segment data blocks of user data are contained within the segment data field of this packet.
- The 64-bit server entity identifier is used to identify the process or group of processes involved in the transaction.
- The 8-bit flag field indicates which fields of the MCB should be interpreted and provides some other optional functions. In particular, MDM and CRE flags indicate whether the message delivery mask field and the coresident entity field are to be interpreted in the packet. The SDA flag indicates whether the segment data field contains data and whether the segment size field should be interpreted.
- The 24-bit request/response is a code set by the user.
- The 64-bit coresident entity identifier identifies an entity or group of entities which must reside on the same host as the server entity.
- The 12-octet user message control block allows user data to be transmitted in the MCB portion of the packet header. This field can be expanded to 20 octets if the MDM and SDA flag bits are zero.
- The 32-bit message delivery field is a mask indicating which 512-octet segment data blocks are being transmitted with a request packet group. In an acknowledgment packet group the field is interpreted as a mask indicating which 512-octet segment data blocks were received correctly by the server. This mask is used to signal which packets of the request must be retransmitted.
- The 32-bit segment size field indicates the number of octets of segment data contained in this packet group.
- The variable length segment data field contains up to thirty-two 512-octet segment data blocks of user data. The actual amount of user data which can appear here is restricted by the maximum transmission unit of the network service.
- The 32-bit checksum field is a checksum calculated over the whole packet or just the header portion of the packet. The checksum is located at the end of the packet to enable the eventual use of hardware streaming techniques to calculate checksums during packet transmission and reception[4].

### 1.2.3.2. Message Transactions

The basis for VMTP communication is the message transaction. A request message is assembled by a client entity and then transmitted to a server entity. When the server entity receives the client entity's request it assembles and transmits the appropriate response to the client entity. No explicit acknowledgment is required as long as all the packet groups of a request message arrive intact. The response message acts as an implicit acknowledgment

to a request message. Response messages are implicitly acknowledged by the receipt of the next request message from the client entity. As long as no packets are lost, communication can be carried out as a sequence of client request messages followed by server response messages without any explicit acknowledgments required.

Multicast message transactions are supported in VMTP by mapping group entity identifiers onto the IP or raw Ethernet multicast facilities. A client entity transmits a request message intended to query a group of server entities. When the request is received by one of the group of server entities, a response is assembled and transmitted to the client entity. VMTP's multicast facility is only a "best effort" multicast, meaning that the client entity only needs to receive a single response regardless of the number of server entities in the group. When the client entity transmits its next request message any further responses from the previous multicast request are ignored.

VMTP also supports streaming of request messages in order to allow efficient implementation of file transfer and other stream oriented applications.

#### 1.2.3.3. Acknowledgment and Selective Retransmission

When a client transmits a request message or a server transmits a response message it transmits each packet group of the message as one burst of packets. When packets are lost during the burst transmission of a packet group, VMTP provides for explicit positive acknowledgments, time-outs and a simple selective retransmission scheme to fill the gaps in the packet group.

Associated with each client entity and server entity is a management server module. The management server module is responsible for receiving packet groups, accounting for all the segment data blocks within a packet group, managing time-outs relative to the reception of packet groups, managing time-outs relative to the retransmission of request

messages and mapping entity identifiers to network addresses. All management server modules are identified using the same well known entity identifier.

Explicit positive acknowledgments for request and response messages are produced by both client and server management server modules when packets have been lost. The positive acknowledgments take the form of a request message addressed to the management server module entity identifier with a coresident identifier corresponding to the originating client or server entity. Selective retransmission is made possible by initializing the message delivery field of the acknowledgment MCB with the appropriate mask to indicate which packets of the packet group were received correctly.

When the acknowledgment is received by the management server module of the originating client or server entity, the packets which need to be retransmitted can be inferred by examining the message delivery field of the MCB. The appropriate packets are then retransmitted by the management server module of the originating client or server entity. This method provides a quick and simple way of providing selective retransmission on a per packet group basis.

#### 1.2.3.4. Rate Based Flow Control

Because VMTP is not a typical sliding window protocol it does not require traditional buffer management flow control. In addition, the problems of packet overrun at the host interface must be addressed because it is the major cause of packet loss on high speed, low error rate networks. Because of this, VMTP incorporates rate-based flow control in order to enforce on transmitters a maximum packet transmission rate acceptable to receivers. In VMTP, rate-based flow control takes the form of an inter-packet gap which forces a transmitter to pause a predetermined amount of time between the transmission of the packets of a packet group.

When a client entity sends a request message it uses the interpacket gap field of the MCB to inform the server entity of the rate at which it can accept the packets of the response message. The client must guess at the initial rate that the server can accept packets but the scheme is adaptable in response to the number of retransmissions required. If a client entity receives an acknowledgment message from a server entity which requests retransmissions, it can adjust its transmission rate accordingly. Conversely, if a client entity is forced to request retransmissions from a server entity it can place an adjusted inter-packet gap value in the MCB of the acknowledgment message in order to inform the server entity to adjust its transmission rate. A very advantageous part of this scheme is that it not only is effective at eliminating packet overrun problems at the endpoints of the transaction, but also at intermediate routers and gateways.

### 1.2.4. GAM-T-103 Transfer Layer Services

In 1987 the French Ministry of Defense published the GAM-T-103 reference model specification which was the culmination of a joint research effort with Electronique Serge Dassault, a French avionics manufacturer [7, 10]. The GAM-T-103 reference model specification defines an architecture and a service specification for military real-time local area networks (MRT-LAN) used on French military ships and aircraft. The GAM-T-103 document does not define specific protocols for data transfer, but merely defines the architecture of MRT-LAN protocols and the services they provide. In the sections which follow we will discuss both the MRT-LAN architecture and the user services provided by the MRT-LAN.

#### 1.2.4.1. Transfer Layer Architecture

In Figure 1.9 we see a figure illustrating the four layers of the MRT-LAN architecture juxtaposed with the seven layer ISO/OSI reference model architecture. As we

can see, the top three layers of the ISO reference model are not present in the MRT-LAN architecture. Also, the MRT-LAN architecture has combined the network and transport layers of the ISO reference model into a single layer known as the transfer layer. The combination of network and transport layer functions into a single functional module is the key feature of the MRT-LAN architecture. Also, the fact that the user application can have direct access to the full functionality of the transfer layer allows the construction of highly optimized real-time applications without sacrificing the need for standard architectures and services.

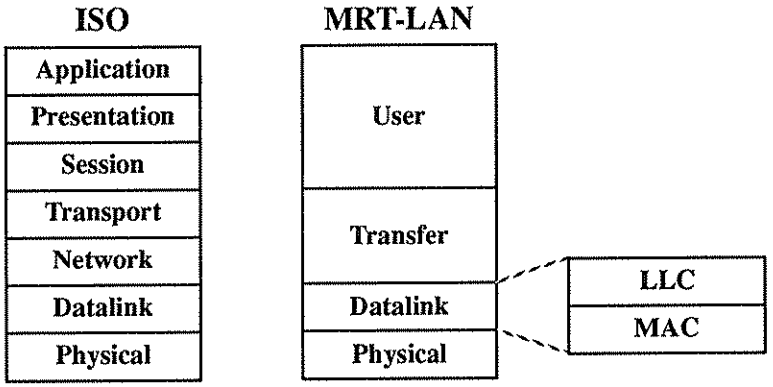


**Figure 1.9** — *MRT-LAN Architecture*

### 1.2.4.2. Transfer Layer Services

There are three types of services provided by the MRT-LAN architecture via the transfer layer. These services are: data communication, synchronization, and management. In the sections which follow this array of offered services will be discussed with emphasis on the data communication services.

#### 1.2.4.2.1. Data Communication Services

The MRT-LAN data communication service specification provides for two classes of connectionless service, classes 0 and 1, and three classes of connection-oriented service, classes 0, 1 and 2. Each type of data communication provides a different level of reliability.

- Class 0 connectionless service provides an unreliable datagram service which can use unicast, multicast, or broadcast transmission. The size of transfer service data units (TSDU) is restricted by the maximum size of the datalink service data units (DSDU).
- Class 1 connectionless service provides a reliable acknowledged datagram service which can use either unicast or multicast transmission. The size of the TSDUs is restricted by the maximum size of the DSDUs.
- Class 0 connection-oriented service provides unreliable delivery of TSDUs with optional duplication control and detection of out-of-sequence data. The size of the TSDUs is restricted by the maximum DSDU size. Unicast, multicast, broadcast and concentration transmission are all available in this service class.
- Class 1 connection-oriented service provides reliable delivery of TSDUs. Correct TPDU sequencing is guaranteed and duplicate TPDUs are eliminated. The size of TSDUs is restricted by the maximum DSDU size. Unicast and multicast transmission are available.
- Class 2 connection-oriented service provides reliable delivery of TSDUs. Correct TPDU sequencing and duplicate elimination is guaranteed. The size of TSDUs is unrestricted.

#### 1.2.4.2.2. Synchronization and Management Services

The synchronization services provided under the MRT-LAN services specification provide two services. First, a global time reference is maintained on the network and can be polled by entities on the network. Second, a teleinterrupt service is provided so that user processes can be signalled or interrupted as a result of events in the network.

The management services allow for the maintenance and monitoring of network entities as well as the detection and configuration of new network entities.

### 1.2.5. The Xpress Transfer Protocol

The Xpress Transfer Protocol (XTP) is being designed and implemented as part of the protocol engine project sponsored by Protocol Engines Incorporated (PEI). The

designer of XTP and co-founder of PEI is Dr. Gregory Chesson, chief scientist at Silicon Graphics Incorporated (SGI).The aim of the protocol engine project is to design a semi-custom VLSI chip set which implements transport and network layer functionality for high speed local, metropolitan, and wide area networks [6]. Figure 1.10 shows a graphical depiction of the protocol engine chip set architecture.

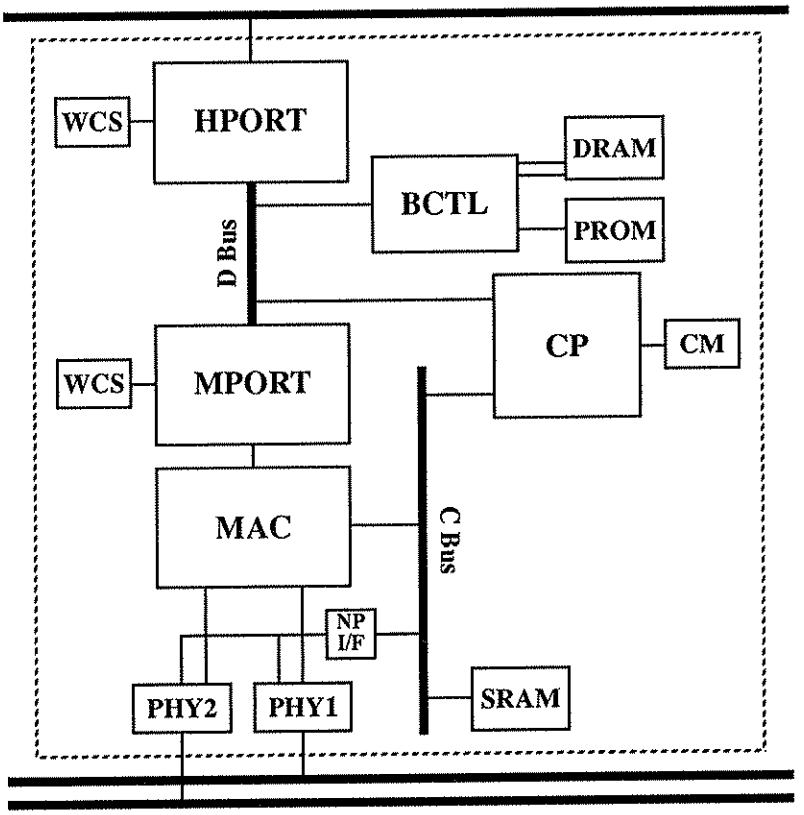


**Figure 1.10** — *Protocol Engine Architecture*

The motivating factor for the protocol engine project is the realization that current transport layer protocols were never designed to support high-speed, low error rate networks like the 100 Mbit/sec Fiber Distributed Data Interface (FDDI) local area network

and were never designed to support the wide array of network applications that now exist as a result of technological advances in network hardware technology. Traditional connection-oriented protocols like TCP and TP4 were originally designed to provide end-to-end reliability services over slow (10 Kbits/sec) and unreliable links. As a result the algorithms of TCP and TP4 were designed with the assumption that transmission errors and lost data were the norm rather than the exception [6]. Also, because the network links were not high performance channels, no mechanisms to handle packet overrun were required; however, on high-speed network links packet overrun is the major source of lost packets [3, 5, 11]. Traditional transport protocols also lack the range of functionality required to support real-time applications and distributed systems.

The design of XTP and many of its services are derived from the French MRT-LAN transfer layer reference model GAM-T-103 [7, 10]. In particular XTP provides a wide range of data communication syntaxes in order to implement both reliable and unacknowledged datagram service and reliable and unacknowledged connection-oriented service. Multicast services are also supported on both an unacknowledged and best-effort basis. XTP also supports selective retransmission, as well as rate and burst control in order to make it suitable for use with high performance networks [16, 17, 19].

#### 1.2.5.1. Navy SAFENET Architecture

Because of XTP's support for many features desirable within a real-time communication system, The United States Navy has selected XTP for use in the Survivable Adaptable Fiber Optic Embedded Network (SAFENET) architecture. The SAFENET architecture is a suite of data communication protocol standards grouped together to provide a data communication system for shipboard mission-critical computer systems [26, 27]. Figure 1.11 shows a profile of the protocols which comprise the SAFENET

architecture. XTP provides the real-time transfer services within the SAFENET architecture.

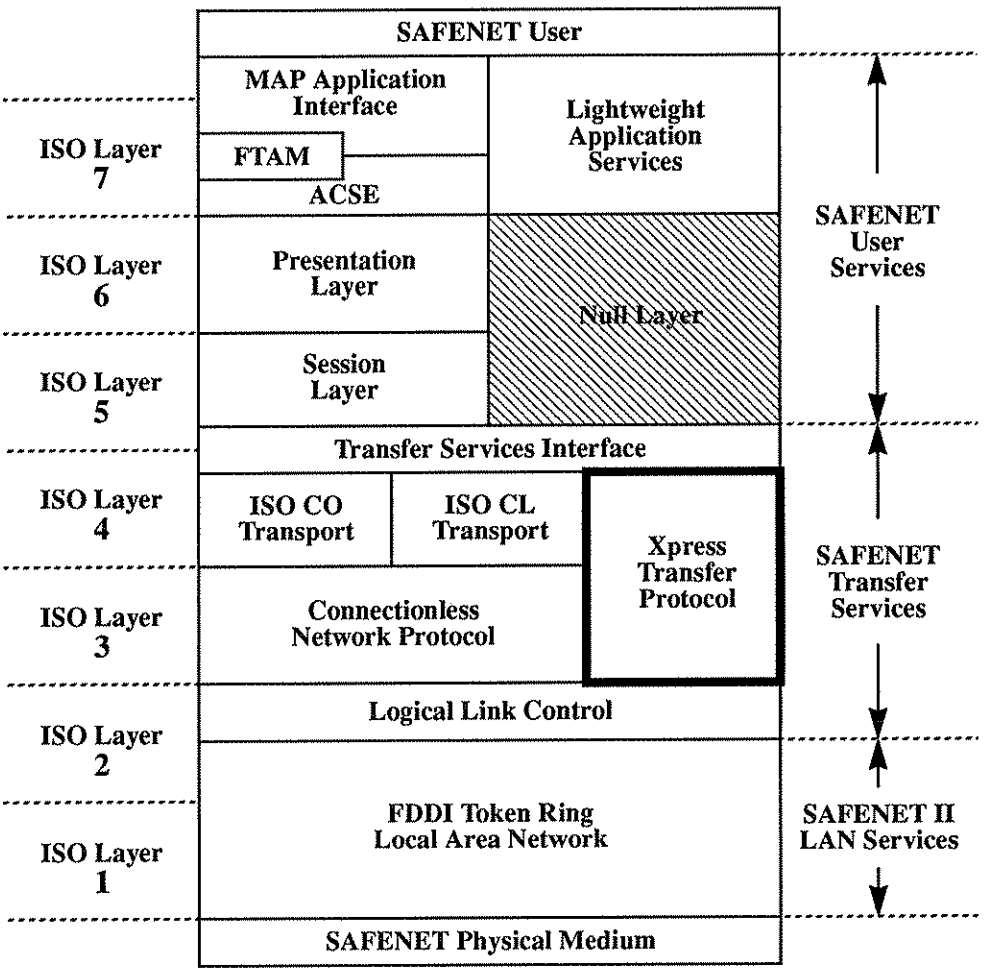


**Figure 1.11** — *SAFENET II Architecture*

Current U.S. Navy shipboard computer systems, like the AEGIS system, use multiple point-to-point links to provide data communication between the individual processing units. The implementation of the multiple point-to-point link system requires special purpose I/O controllers and redundant cabling. Future shipboard computer systems

will require more connectivity to support more complete system integration and consequently will require more I/O controllers and cabling. SAFENET is being developed as a solution to these future system integration problems [27].

In the sections which follow we will discuss the major features of XTP, with emphasis on its packet formats, data transmission syntax, and error control mechanisms. We will also discuss XTP's multicast facility and some of the algorithms used to implement XTP's best-effort multicast service.

#### 1.2.5.2. Packet Formats

XTP features a header/trailer packet format designed specifically to allow efficient hardware pipelining. The 24-octet header and 16-octet trailer have a fixed format and all XTP packets are required to be aligned on a 8-octet boundary. The middle segment of the XTP packet is either an information segment or a control segment depending on the packet type. Figure 1.12 shows a breakdown of the fields of the XTP headers, trailers and middle segments. We will briefly discuss the fields of each portion of the XTP packet.

**XTP Header Fields**

- The 4-octet *cmd* field contains a 2-octet *options* field, a 1-octet *offset* field, and a 1-octet *type* field. The *options* field is a set of flags which indicate whether a variety of options are in effect for this packet. These options include: a flag indicating whether the packet header and trailer are in little endian or big endian byte order, whether the middle segment checksum is to be calculated, and whether the packet is a multicast packet. The *offset* field indicates the number of padding bytes immediately preceding an information segment. The *type* field indicates the packet type.
- The 4-octet *key* field is used as a short form address to identify the context to which the packet belongs.
- The 4-octet *sort* field is used to implement priority service within XTP. It is optionally interpreted based on a flag in the *options* field.
- The 4-octet *seq* field contains the sequence number of the packet sender's half of the context's full duplex data stream. For *First* and *Data* packets the *seq* field represents the sequence number of the first non-offset padding octet in the information segment. For *Cntl* and *Path* packets the *seq* field represents the next octet to be transmitted on the packet sender's data stream.

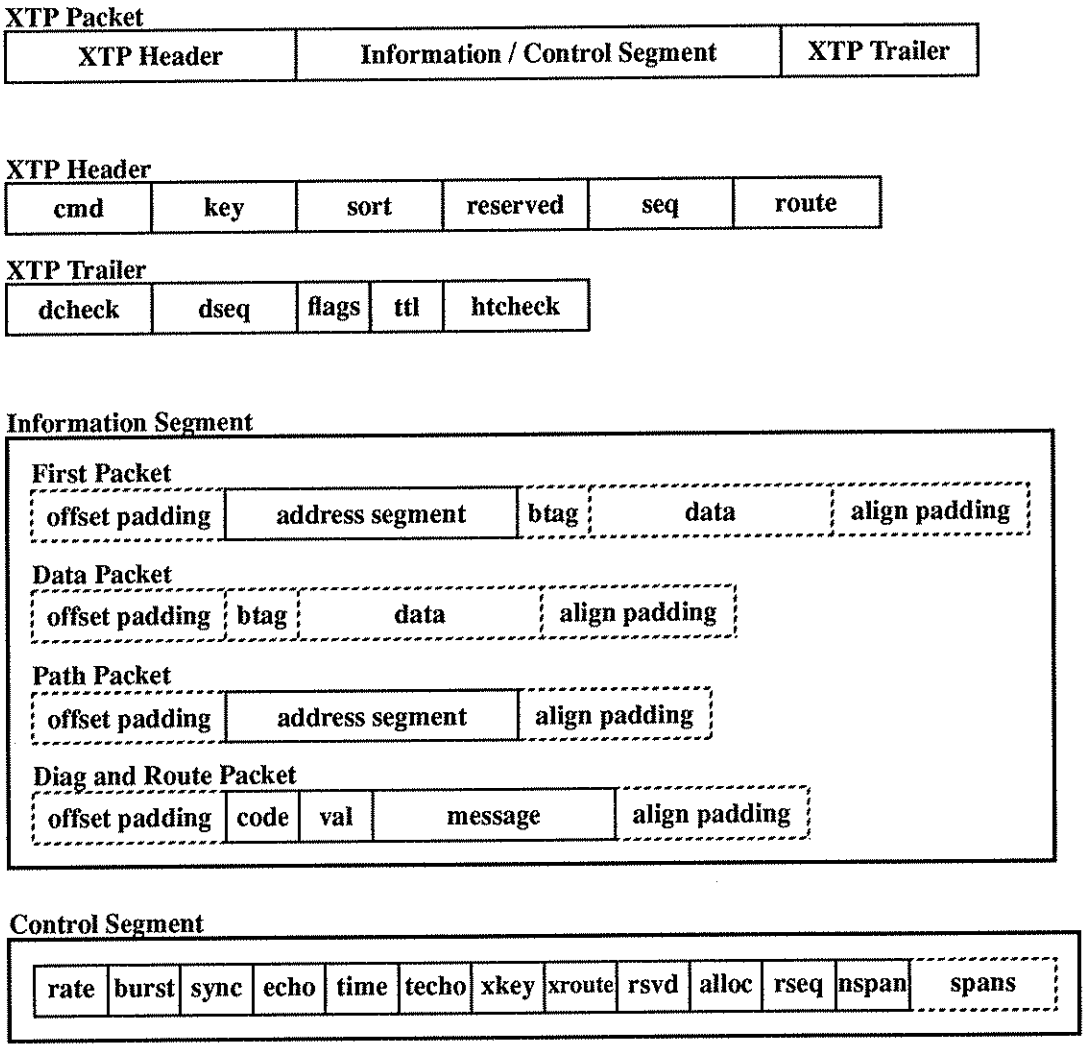


**Figure 1.12** — *XTP Packet Formats*

- The 4-octet *route* field is used as a short form address for identifying a path through one or more XTP switches.

## XTP Trailer Fields

- The 4-octet *dcheck* field is a checksum calculated over the entire middle segment of the XTP packet.

- The 4-octet *dseq* field is the sequence number value which can have two interpretations. In the first case it is used by the receiver to indicate to the sender the sequence number of the next octet of data to be delivered to the client. In the second case it is used to initialize sequence numbers during connection establishment.
- The 10-bit *flags* field is used to issue commands to the XTP receiver. Two of the flag bits are of particular interest. The SREQ bit is used by the XTP sender to request an immediate response from the XTP receiver in the form of a *Cntl* packet. The DREQ bit is used to request a response when the data associated with the current packet's *seq* value is delivered to the user application.
- The 2-octet *ttl* field is interpreted as a packet's time to live. Each tick of the field represents 1 ms.
- The 4-octet *htcheck* field is a checksum calculated over the XTP header and trailer.

**XTP Information Segment**

- The variable length *offset padding* field is used to help align data on required 8 byte boundaries and meeting the minimum packet length requirements of underlying protocols.
- The variable length *align padding* field is used to align data on the required 8 byte boundary.
- The variable length address segment is used in *First* and *Path* packets to convey network and MAC layer addresses to the receiver. XTP supports a number of network addressing formats including IP addresses and ISO network layer addresses.
- The optional 8-octet *btag* field is used to transport out-of-band data. Tagged data is included in the sequence space but is not user data. This field is intended for use by higher layer applications to pass control information.The presence of *btag* data is signalled by the setting of a flag in the header.
- The variable length data segment is used for carrying user data. An empty *Data* packet is simulated by inserting a specially encoded *btag* field.
- The 4-octet *code* field is used by *Diag* and *Route* packet types to specify the type of message contained in the *message* field.
- The 4-octet *val* field is used by the *Diag* packet type to provide subcodes to the *code* field.
- The variable length *message* field is intended for upper layer application use. Its contents are not interpreted.

**XTP Control Segment**

- The 4-octet *rate* field is used by a receiver to indicate to a transmitter how much data the receiver is prepared to receive in a particular length of time. In this case rate is expressed in units of octets per second.
- The 4-octet *burst* field is used by a receiver to indicate to a transmitter how much data the receiver is prepared to receive in one transmission burst.

- The 4-octet *sync* and *echo* fields are used for protocol state machine synchronization. The *sync* values received on incoming *Cntl* packets are transmitted back in the *echo* field of outgoing *Cntl* packets.
- The 4-octet *time* and *techo* fields are used in the measurement of round trip time. The *time* value received with incoming *Cntl* packets is transmitted back in the *techo* field of outgoing *Cntl* packets.
- The 4-octet *xkey* and *xroute* fields are used for key and route exchange.
- The 4-octet *alloc* field is used by receivers to implement flow control. The transmitter may send data up to but not including the value of the *alloc* field.
- The 4-octet *rseq* field indicates the sequence number of the next octet of data expected by a receiver.
- The 4-octet *nspans* field indicates the number of spans contained in the *spans* field.
- The *spans* field contains pairs of 4-octet sequence numbers. These sequence number pairs represent spans of the sequence space which have been correctly received by the receiver.

#### 1.2.5.3. Data Transmission

XTP is very much like TCP in that it provides a full-duplex octet stream between two connection endpoints or contexts. Sequence numbers are assigned on a per octet basis and each half of the full-duplex stream has its own sequence space. XTP, however, provides a much richer set of data transmission syntaxes in order to support transport services other than reliable connection-oriented service. The ability of the XTP *First* packet to carry user data, and the design of the XTP receiver to operate as a slave of the XTP sender, gives XTP the flexibility to provide a number of different transport services.

In addition to full-duplex connection-oriented service, XTP supports transport level datagram service using both a three-way handshake and a fast two-way handshake. Using the End-of-Message (EOM) bit in the trailer *flags* field allows the implementation of multi-packet message systems for use in distributed systems applications. Figure 1.13 illustrates three possible types of XTP interaction.

For real-time applications where XTP is being used to transport frequently repeated data which does not require reliable transmission, XTP provides the ability to disable error

control at the receiver. By setting the NOERROR bit in the *options* field of the XTP header the sender informs the receiver to disable error control. Any *Cntl* packets transmitted from the receiver to the sender will indicate that all data has been passed to the user regardless of whether or not the data arrived correctly.

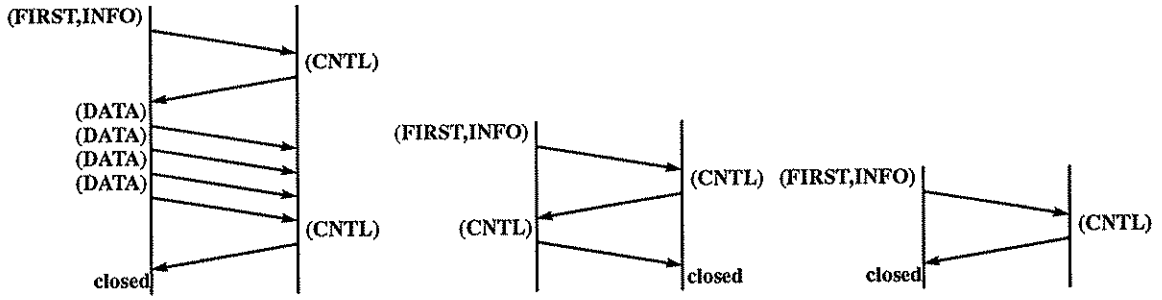


**Figure 1.13** — *Data Transmission Syntaxes*

### 1.2.5.4. Acknowledgments and Retransmission

An XTP receiver has no timers and takes very little independent action with respect to acknowledging data transmitted by the sender. In particular, if only simplex communication is occurring, acknowledgments, in the form of *Cntl* packets, are transmitted by the receiver only at the request of the sender. The sender makes acknowledgment requests by the use of the SREQ and DREQ bits in the XTP trailer's *flags* field. In full-duplex data communication the *dseq* field of incoming *Data* packets is used to acknowledge data which has been delivered to the user, allowing buffer space to be freed by the sender.

XTP *Cntl* packets received in response to SREQ or DREQ *Cntl* packets contain four fields of interest. The *alloc* field contains a value one larger than the largest sequence value that the receiver has buffer space to accept. The *rseq* field contains a value one larger than

the largest monotonic sequence number received without error. The *nspans* field contains the number of correctly received spans of data which lie between *rseq* and *alloc* and the *spans* field contains pairs of sequence numbers corresponding to the correctly received spans of the sequence space. Figure 1.14 illustrates the relationship between these fields. The spans are used by the receiver to indicate what data needs to be selectively retransmitted.

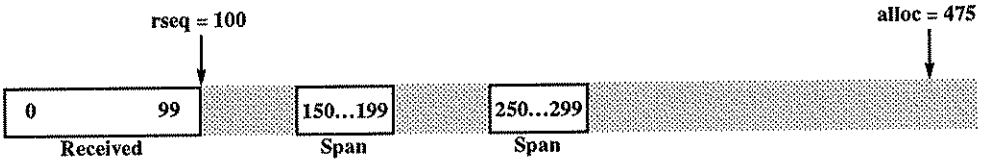


**Figure 1.14** — *Rseq, Alloc and Spans*

### 1.2.5.5. Multicast Transmission

Like VMTP, XTP provides a multicast facility. XTP's multicast facility can be operated without error control by setting the NOERROR bit, or it can be operated in a best effort fashion using *rseq* values furnished by the receivers in *Cntl* packets to implement a go-back-n retransmission strategy. This is only a best-effort multicast because at some point, if a receiver falls too far behind, it will simply drop out of the multicast conversation. XTP specifies a procedure known as the bucket algorithm to provide multicast receivers with a window of opportunity to respond to a sender request for a *Cntl* packet. The bucket algorithm will be discussed in more detail in the following subsection. XTP multicast receivers also implement a procedure for suppressing redundant control packets. This procedure is known a slotting and damping, and will be discussed in section 1.2.5.5.2.

#### 1.2.5.5.1. The Bucket Algorithm

The bucket algorithm is a procedure implemented by XTP to allow a multicast sender to accumulate data about the state of the multicast receivers over a specified time period and then, after that time period has expired, apply the accumulated data to the state of the sender.

XTP multicast senders periodically request acknowledgments from multicast receivers by transmitting a *Cntl* packet with the SREQ bit set in the trailer. When the *Cntl* packet is transmitted the multicast sender accumulates the most conservative values contained in the responses from the multicast receivers in a bucket. Buckets are uniquely identified by the *sync* value sent out in the SREQ *Cntl* packet and response *Cntl* packets are associated with buckets by their *echo* value. The values accumulated in each bucket are the minimum *alloc*, *rseq*, and *dseq* values and the maximum *rtt* value over all responses associated with the bucket.

A multicast sender may have multiple buckets which are recycled periodically. Specifically, when a multicast sender needs to send a SREQ *Cntl* packet to the multicast receivers, it applies the values contained in the oldest bucket to its state and then the oldest bucket becomes the current bucket. A small number of buckets allows the values contained in the buckets to be applied to the sender's state more quickly and increases throughput. However, a small number of buckets provides the multicast receivers with a small time window in which to respond and can lead to slowly responding receivers being dropped from the conversation. A large number of buckets provides a larger time window for multicast receivers to respond and allows slowly responding receivers to maintain conversation membership at the expense of performance. In this way XTP's multicast facility can be tailored to the characteristics of the multicast conversation participants and the reliability requirements of the multicast conversation.

#### 1.2.5.5.2. Slotting and Damping

When an XTP multicast sender requests an acknowledgment from the multicast receivers, the request is made without knowledge of the number or configuration of the stations of the multicast group. Since receivers must respond immediately to sender SREQ *Cntl* packets, there exists the possibility that for large homogenous multicast groups the network could experience bursts of *Cntl* packet traffic. To combat this phenomenon XTP multicast receivers implement procedures for spacing responses to SREQ *Cntl* packets over time (slotting) and suppressing redundant responses (damping).

Slotting is implemented by assigning each multicast receiver a random slot time based on some local seed such as the receiver's MAC address [17]. The allotted time slot is used by the receiver to determine how long after receiving an SREQ *Cntl* packet the receiver should wait before transmitting a *Cntl* packet in response. In this way the *Cntl* packets transmitted by the multicast receivers can be spaced over some period of time to help avoid an avalanche of *Cntl* packets on the network.

Damping is used by multicast receivers to prevent their redundant or outdated multicast *Cntl* packets from being transmitted. Multicast receiver's *Cntl* packets are multicast to all members of the multicast group. When a multicast receiver receives a *Cntl* packet from another multicast receiver it examines the packet to see if its own pending *Cntl* packet contains information which is redundant. If its *Cntl* packet is redundant then the multicast receiver suppresses the transmission of the packet and allows the packet it received from the other multicast receiver to provide the multicast sender with appropriate information.

Slotting and damping working in concert provide a mechanism to smooth XTP multicast *Cntl* packet traffic and also reduce the amount of multicast *Cntl* packet traffic.

#### 1.2.5.6. Flow Control

The flow control mechanism used in XTP is similar to that of other stream-based transport protocols. The *alloc* field in control packets is used by receivers to indicate to senders an upper limit on sequence numbers which should be transmitted. The relationship $dseq \leq rseq \leq alloc$ (modulo $2^{32}$) describes the relative positions of *dseq*, *rseq*, and *alloc* in the sequence space.

#### 1.2.5.7. Rate and Burst Control

In addition to traditional flow control mechanisms XTP provides rate and burst control which are used to help alleviate packet overrun problems. Rate and burst control is of particular importance in XTP because of the potential problems created by having high-speed VLSI implementations of XTP attempting to communicate with slower software implementations.

XTP receivers advertise their rate and burst requirements by using the *rate* and *burst* fields of the *Cntl* packets. When a connection is opened the initiator uses default values for *rate* and *burst*. When the receiver responds with a *Cntl* packet its *rate* and *burst* requirements are contained in the *cntl* segment.

The *rate* value is the number of octets per second that the receiver is capable of accepting. The *burst* value is the maximum number of bytes per burst of packets that the receiver is capable of accepting. By dividing the *rate* value by the *burst* value we get a value equal to the number of packets per second the receiver is prepared to receive. This, coupled with a timer whose period is *burst* divided by *rate*, forces the proper time spacing between packet bursts. When the *rate* value is 0 all rate control is disabled. When the *burst* value is 0 transmission is halted.

## 1.3. Conclusion

The balance of this thesis will focus on the design and performance of a software implementation of XTP built by the staff of the University of Virginia's Computer Networks Laboratory. The performance analysis will focus on many of the features of XTP previously discussed and will include relevant comparisons to a TCP implementation running on the identical hardware platform.

*Chapter 2*

# A Software Implementation of XTP

## 2.1. Introduction

In 1988 the University of Virginia's Computer Networks Laboratory began work on a software implementation of the XTP under the sponsorship of Sperry Marine Incorporated. Sperry's sponsorship was motivated by their general interest in the emerging SAFENET architecture and their desire to have a seamless transition from their own SeaNET shipboard communication system to the SAFENET architecture. SeaNET was originally developed by the UVa Computer Networks Laboratory and later commercialized by Sperry Marine for use in their integrated bridge systems for commercial ships.

Since the delivery of the original XTP version 3.4 implementation to Sperry Marine in the spring of 1990, the staff of the UVa Computer Networks Laboratory has continued upgrading the original implementation to keep it current with the most recent release of the XTP protocol specification. The upgrade process has included adding support for Ethernet and porting the implementation from its original Intel 80x86 IBM PC/AT Token Ring based implementation, to a Motorola 68020 processor using an FDDI MAC and VME backplane bus. The version currently supported on the Intel 80x86 PC/AT platform is XTP version 3.5.

## 2.2. Hardware Environment

Because Sperry Marine's SeaNET based integrated bridge system was developed using an IBM PC/AT hardware platform, Sperry funded the development of XTP for the same hardware platform. The current version of XTP runs on IBM PC/AT compatible computers using either an Intel 80286 or 80386 processor.

The UVa Computer Networks Laboratory has a variety of Intel 80386-based machines which vary in the processor and bus clock speed. Table 2.1 lists the four basic classes of machines employed by the Laboratory in the development of XTP 3.5. The machines in the table are listed in descending order based on their performance characteristics. The CORE International machines with the 25 MHz processor clock and the 12.7 MHz bus clock are the fastest machines available in the Laboratory. Because of the faster bus clock the CORE International machines provide a significant performance gain when copying data to and from devices attached to the bus.

| Class | Name | CPU | Bus |
|---|---|---|---|
| 1 | CORE International Turbo AT | 25 MHz 80386 | 12.5 MHz AT-Bus |
| 2 | ALR FlexCache 25386X | 25 MHz 80386 | 8.3 MHz AT-Bus |
| 3 | Zenith Z-386/20 | 20 MHz 80386 | 8.3 MHz AT-Bus |
| 4 | Zenith Z-386/16 | 16 MHz 80386 | 8.3 MHz AT-Bus |

**Table 2.1** — *Machine Classifications*

## 2.3. MAC Layer Hardware Environment

The IBM PC/AT implementation of XTP currently supports two MAC interfaces. Standard 10 Mbps baseband coaxial Ethernet is supported using the Western Digital WD8003E EtherCard Plus interface [30]. The IEEE 802.5 Token Ring is supported using the Proteon p1340 4 Mbps token ring interface and the Proteon p2700 multi-station wire center [29]. In the following two sections we will discuss some of the features of these MAC interfaces with emphasis on those features which have ramifications with respect to performance.

### 2.3.1. Western Digital WD8003E Ethernet Interface

The Western Digital WD8003E Ethernet interface has three sets of control registers which can be accessed using the 80x86 port I/O instructions. These registers allow for the configuration and operation of the National Semiconductor DP8390 Ethernet chip set.

The key feature of the WD8003E is the on-board 8-Kbyte dual ported RAM buffer. This buffer can be mapped directly into the machine's address space so that data can be copied directly into the buffer. This feature allows packets to be copied on and off the board using simple 80x86 move instructions rather than more costly I/O instructions or DMA. The buffer is divided into thirty-two 256-byte blocks. After setting aside six blocks for a transmission buffer we have 26 blocks left to buffer incoming packets. This feature makes this board capable of handling large bursts of packets and has allowed the board transmitter and receiver to be fairly well matched in terms of performance.

### 2.3.2. Proteon p1340 Token Ring Interface

The Proteon p1340 token ring interface utilizes the Texas Instruments TMS380 chip set to implement the token ring MAC layer functions. This chip set has a programming interface; however, it operates on a very low level. It requires the device driver to recognize and handle correctly the wide array of control packets used by the IEEE 802.5 token ring MAC protocol.

The alternative to using the low level MAC interface provided by the TMS380 chip set was to utilize the LLC protocol firmware contained in the p1340's EPROMs. The LLC firmware handled the complexities of the low level MAC interface while providing a programming interface which allowed access to the MAC interface's data transmission and reception capabilities. Because the full LLC functionality was not required the LLC

programming interface was placed in promiscuous mode so that all properly addressed MAC frames would be received by the LLC firmware.

The LLC programming interface is controlled using a set of control registers, and a set of control blocks and packet buffers allocated in user memory. Commands are issued to the interface by formatting the appropriate control block and then using a port I/O instruction to inform the interface of the available command. The interface then uses DMA to transfer the control block to the interface's on-board memory. The movement of packets on and off the board is handled in a similar fashion.

Experience with the Proteon p1340 LLC programming interface indicates that the receiver is significantly slower than the transmitter. This has some interesting consequences with respect to the performance of the XTP 3.5 implementation. These consequences will be discussed in more detail in Chapter 3.

### 2.4. UVa XTP 3.5 Software Architecture

The UVa Computer Networks Laboratory's software implementation of XTP 3.5 is implemented as five light-weight tasks grouped into three task sets. The tasks operate on two sets of shared data structures [9, 12, 20]. Figure 2.1 shows how the light-weight tasks are grouped

The device processor contains two tasks—the MAC task and the LLC task. The MAC task provides the software services to the MAC hardware interface. The LLC task is available to implement the logical link control protocol of choice. No LLC protocol is currently implemented in UVa XTP 3.5.

The context processor provides the XTP protocol processing for up to thirty-two simultaneously open contexts. The context processor maintains the protocol state information about each open context. The context processor is also referred to as the XTP

engine. The interaction between the context processor and the device processor uses request/response semantics and shared data structures known as frame structures.

The XTP drivers are divided into two tasks—the user task and the timer task. The user task executes the user application code. The timer task handles most of the required XTP timer functions and events.

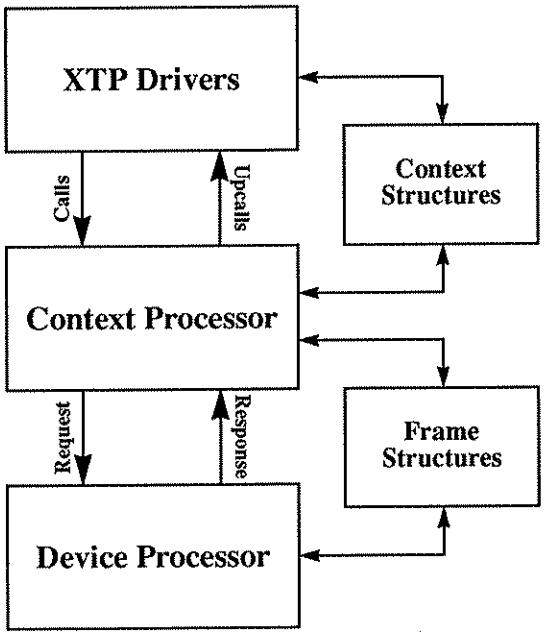


**Figure 2.1** — *UVa XTP Architecture*

In the next section we briefly describe the light-weight task scheduler used to implement the UVa XTP 3.5 architecture's multi-tasking.

### 2.4.1. ZEK — Fast Real-Time Scheduler

The host operating system for the UVa XTP 3.5 is MS-DOS. Since MS-DOS is a single threaded operating system it was necessary to implement a real-time scheduler in

order to provide the multi-tasking required by the UVa XTP 3.5 architecture. The scheduler implemented is called ZEK [9, 12, 20].

ZEK is a preemptive, light-weight task scheduler with sixteen priority levels. It has a context switch latency of approximately 25 μs on a class 1 or a class 2 machine. It is implemented as a Turbo C large model library which is linked to the UVa XTP application code. The only portions of the MS-DOS operating system used by UVa XTP 3.5 are the file system and the loader.

| ZEK Priority | UVa XTP Task |
|---|---|
| 0 | King Processor Task |
| 1 | Null Task |
| 2 | MAC Processor Task |
| 3 | LLC Processor Task |
| 4 | Null Task |
| 5 | Context Processor Task |
| 6 | DOS Processor Task |
| 7 | Client Processor Task |
| 8 | Timer Processor Task |
| 9 | User Processor Task |
| 10 | Null Task |
| 11 | Null Task |
| 12 | Null Task |
| 13 | Null Task |
| 14 | Null Task |
| 15 | Idle Processor Task |

**Table 2.2**—*XTP Task Priorities*

Preemption in ZEK can only occur when an interrupt service routine, initiated as the result of an external event, marks a higher priority task ready to run. In all other cases the currently executing task must explicitly surrender the processor to another task. Only

one process can operate on each priority level. Table 2.2 lists the priority level of the UVa XTP 3.5 tasks.

### 2.4.2. XTP Engine and Driver Interaction

Interaction between the XTP drivers and the XTP engine is implemented via a set of function calls, function upcalls and shared data structures known as context structures. A context structure contains all the state variables for a particular open context. The key portions of the context structure are the context ring buffers. Each half of a context's full-duplex channel has a pair of ring buffers associated with it. These ring buffers are known as the data ring and the event ring. These rings allow inbound and outbound buffering for two communication channels, the data channel and the event channel [9, 12, 20]. Figure 2.2 shows both transmit and receive ring buffer structures with their associated pointers.

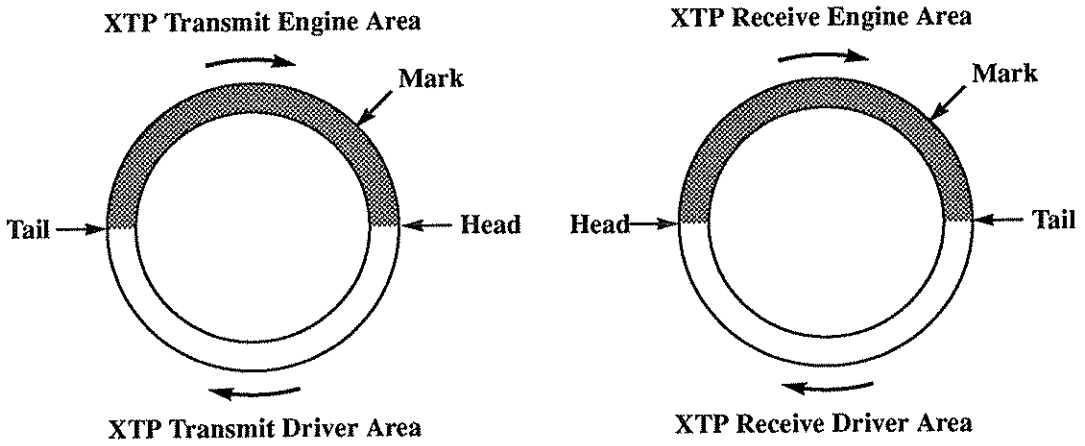


**Figure 2.2** — *XTP Ring Buffers*

The head pointer points to the last item written into the ring. For receive rings the XTP engine moves the head pointer; whereas for transmit rings the XTP driver moves the

head pointer. The tail pointer points to the next item to read from the ring. For transmit rings the tail pointer is moved by the XTP engine and for receive rings it is moved by the XTP driver.

The mark pointer is maintained exclusively by the XTP driver. It functions as a synchronization point between the XTP engine and the XTP driver. In the transmit ring case, when the tail pointer passes the mark pointer the XTP driver is signalled using a predefined upcall routine. In the receive case a signal is issued when the head pointer passes the mark pointer.

### 2.4.3. XTP Protocol Engine Drivers

Managing the interaction between the XTP engine and the XTP drivers is extremely difficult and error prone. It requires not only a full knowledge of the XTP protocol specification but also knowledge of the implementation details of the XTP engine and the XTP device processor. Because of this problem, a standard set of XTP drivers has been implemented as a set of library routines to allow application programs to be written quickly.

The XTP driver library functions were designed to provide services similar to the services provided by the C programming language standard I/O library. Based on the type of service provided these functions can be divided into four groups — resource allocation, memory data transfer, character-oriented data transfer, and device-oriented data transfer. These four groups of functions will be discussed in the following sections [12, 20].

#### 2.4.3.1. Resource Allocation and Deallocation Drivers

The function `X_open()` is used to allocate and initialize a context structures. However, it does not attempt to open a connection. It merely binds the context to a particular destination address. This allows XTP drivers to take full advantage of the XTP *First* packet's ability to carry user data. The `X_open()` function returns a `XFILE` number

which acts as the handle for the open context. It is analogous to the file descriptor returned by the UNIX system call `open()`.

The `X_close()` function takes as an argument an `XFILE` number and closes the appropriate connection and deallocates the context structures associated with that connection.

#### 2.4.3.2. Memory Transfer Drivers

The functions `X_putb()` and `Xputm()` are used the transmit a single byte of data or a block of data respectively. Depending on the arguments supplied these functions can implement blocking, partial-blocking, or non-blocking semantics.

The functions `X_getb()` and `X_getm()` allow the reception of a single byte or a block of data respectively. Function `X_getb()` implements non-blocking semantics and can be used to pole for the availability of data. Function `X_getm()` blocks until the buffer supplied as an argument is filled with data.

#### 2.4.3.3. Character-Oriented Drivers

The functions `X_putc()` and `X_print()` are used to transmit a single character of data or a formatted string of data respectively. All data produced by calls `X_putc()` and `X_print()` is buffered until a newline character appears in the character stream or immediate transmission is requested using the `X_flush()` function.

Function `X_getc()` requests a character from the incoming character stream. The `X_getc()` function implements blocking semantics. The `X_ungetc()` function forces a character back into the incoming character stream. This function only operates within a single message boundary.

#### 2.4.3.4. Device-Oriented Drivers

The device-oriented drivers allow data from a byte or block oriented device to transferred directly to the network and data from the network to transferred directly to such a device. This interface provides a faster way to implement file transfers because it avoids an intermediate copy.

Function `X_putf()` transmits a file over the network using the standard `read()` system call to copy data from the file directly to the transmit data ring. The `X_getf()` function receives a file over the network using the standard `write()` system call to copy the data from the receive data ring to the file.

*Chapter 3*

# Performance Measurements and Analysis

## 3.1. Introduction

Our performance measurements of the software implementation of XTP 3.5 developed at the UVa Computer Networks Laboratory focused on three basic performance metrics — throughput, latency and delay. Specifically, we were interested in the throughput, latency and delay performance that a user of the high level XTP memory transfer driver would experience. All our experiments were based on four short application programs using the allocation and deallocation routines `X_open()` and `X_close()`, and the memory transfer routines `X_putm()` and `X_getm()`. The experiments were performed using single segment 802.5 Token Ring and 802.3 Ethernet LANs installed in the UVa Computer Networks Lab.

### 3.1.1. Timers

All timing required for the experiments was implemented by using the IBM PC/AT system timer. The IBM PC/AT system timer is a 16-bit clock that operates at 1.190 MHz. This gives us a timer period of approximately 0.84 μs, however, it requires at least 10 μs to poll the clock using 80x86 I/O instructions on a 25 MHz 80386 machine. This gives us an effective resolution of no better than 10 μs. The system timer can also be set to issue periodic interrupts. For experiments which did not require a high resolution timer, we set the system timer to interrupt every 65535 clock ticks, or approximately every 55 ms. The elapsed time was deduced from the number of timer interrupts which occurred during an experiment.

### 3.1.2. Throughput Measurements

All throughput measurements are expressed in terms of user data bits reliably transferred per second. All packet overhead associated with the MAC protocol header and the XTP header and trailer is ignored in these measurements. All measurements were obtained by measuring the elapsed time required to transfer 2 Mbytes of user data. All experiments were performed at least twice and then averaged to obtain a single data point. The experiments were parameterized using message lengths between 16 bytes and 16 Kbytes.

### 3.1.3. Latency Measurements

For the purposes of our experiments we defined two different latency metrics. These metrics will be referred to as *round-trip* latency and *one-way* latency. *Round-trip* latency is defined to be the amount of time required to send a message and receive its acknowledgment from the receiving context. *One-way* latency is defined to be half the amount of time required to send a message to a receiving context and receive the same message back from the receiver. We used the *one-way* latency metric when comparing XTP with TCP/IP. Each latency measurement was performed at least ten times and then the average of the measurements was taken. The experiments were parameterized by message lengths between 16 bytes and 4 Kbytes.

### 3.1.4. Delay Measurements

The delay metric is defined to be the amount of time required for the transport layer to fulfill a sequence of service requests. For example, we might be interested in the amount of time required to establish an XTP or TCP/IP connection. The delay metric is distinct from either of the latency metrics in that the delay metric is attempting to measure the

combination of time delay incurred by local overhead as well as latency from actual data transmission. All delay measurements were averaged over at least ten samples.

## 3.2. XTP Unicast Measurements

Our first set of experiments focused on the unicast performance of UVa XTP 3.5. Specifically, we were interested in determining the throughput and *round-trip* latency characteristics of both the XTP engine and the XTP memory transfer driver. By measuring the throughput and *round-trip* latency of both the low level interface and high level driver interface we could draw some conclusions about the efficiency of the high level interface. All the unicast experiments used class 1 machines (see Table 2.1) as both transmitter and receiver.

### 3.2.1. IEEE 802.5 Token Ring Implementation

#### 3.2.1.1. Throughput Measurements

In Figure 3.1 we see a graph which plots message size on the horizontal axis versus throughput on the vertical axis. This graph indicates that the maximum throughput of the XTP memory transfer driver is just over 2.2 Mbits/sec when using 16 Kbyte messages. In addition, the XTP low level interface and the XTP memory transfer driver have virtually identical throughput performance.

Figure 3.2 illustrates the relative efficiency of the XTP memory transfer driver versus the XTP low level interface. The graph expresses the performance of the memory transfer driver as a percentage of the performance of the low level interface. The graph in Figure 3.2 indicates that the XTP memory transfer driver performance has exceeded the performance of the XTP low level interface for some message sizes. On average the memory transfer driver deliveries slightly over 100% of the throughput performance delivered by the low level interface. The implication of Figure 3.2 is that the throughput

performance bottleneck for the 802.5 Token Ring implementation of XTP is not the XTP memory transfer driver or the XTP engine, but rather the Proteon p1340 MAC processor. There are two scenarios which could cause this behavior.

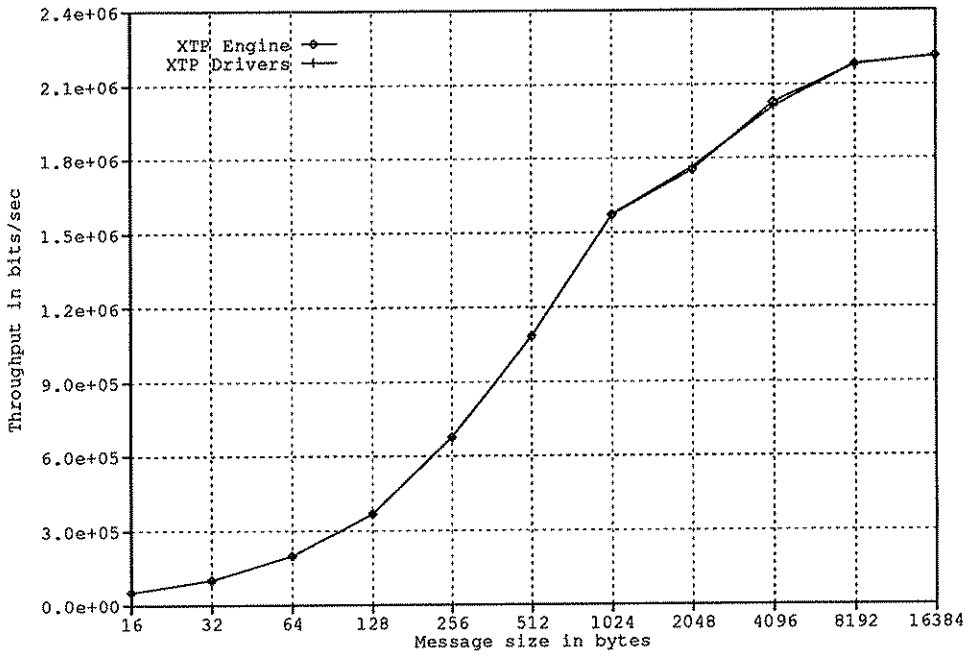


**Figure 3.1** — *XTP Unicast Throughput on Token Ring*

First, because the Proteon p1340 receiver is slower than the transmitter, the extra layer of delay added by the XTP driver allows the receiver to keep better pace with the transmitter. If the receiver is keeping up with the transmitter, then fewer packets will be dropped leading to fewer retransmissions. Fewer retransmissions implies better throughput performance.

Second, our experience with the transmitter of the Proteon p1340 interface indicates that the MAC transmission interface is likely to be the bottleneck in the system rather than

the XTP drivers or XTP engine. An older version of the Proteon p1340 MAC processor had a raw transmission throughput of 1.8 Mbits/sec [20]. The Proteon p1340 MAC processor has been improved significantly since then; however, relatively slow DMA is still used exclusively for the movement of data between the host and the interface. As a result the cost of setting up and waiting for DMA to complete causes the MAC device processor to be the bottleneck in the system.

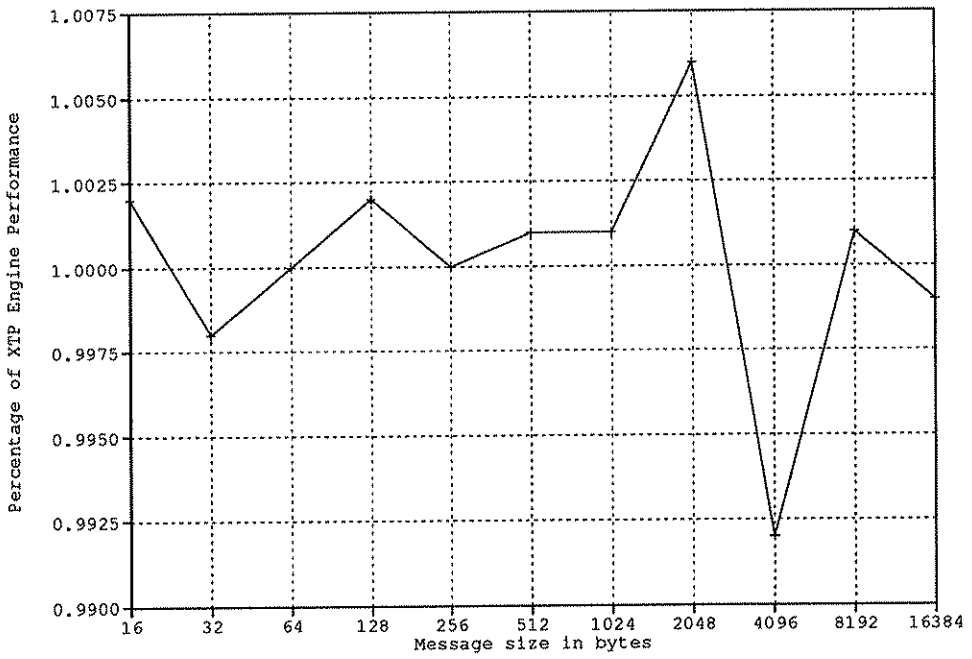


**Figure 3.2** — *XTP Driver Throughput Efficiency on Token Ring*

### 3.2.1.2. Round-trip Latency Measurements

In Figure 3.3 we see a graph which plots message size on the horizontal axis versus *round-trip* latency on the vertical axis. For a message of 16 bytes the *round-trip* latency which can be expected from the XTP memory transfer driver is 4.92 ms. As can be seen the *round-trip* latency curve is nearly linear with message size.

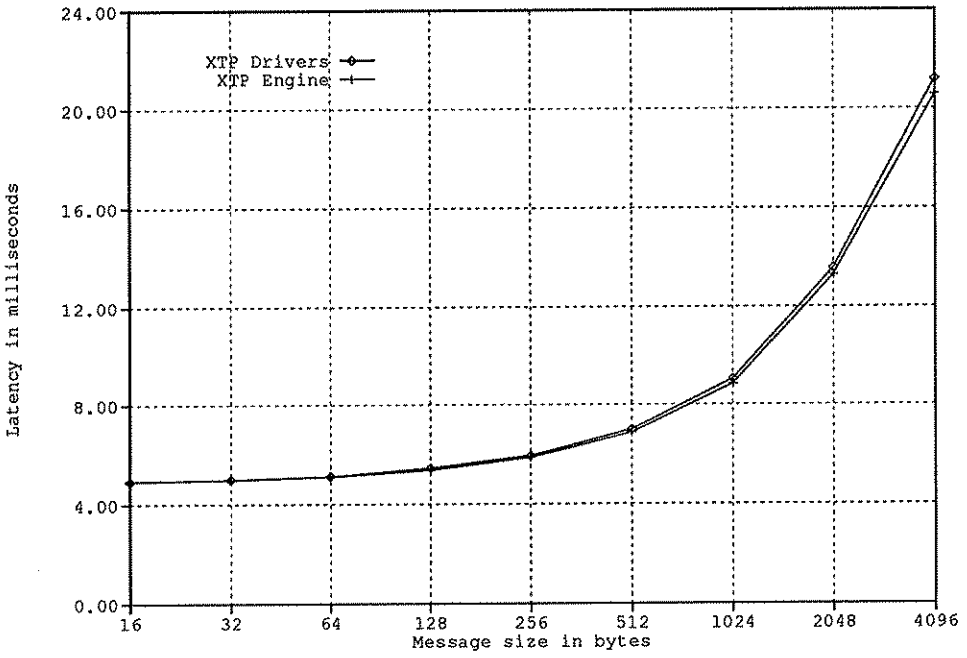


**Figure 3.3** — *XTP Unicast Round-trip Latency on Token Ring*

Like Figure 3.2, Figure 3.4 provides a measure of the efficiency of the XTP memory transfer driver measured as a percentage of the *round-trip* latency provided by the XTP low level interface. Averaged over all message sizes, the memory transfer driver has *round-trip* latency performance only 1% worse than the *round-trip* latency performance received from the low level interface.

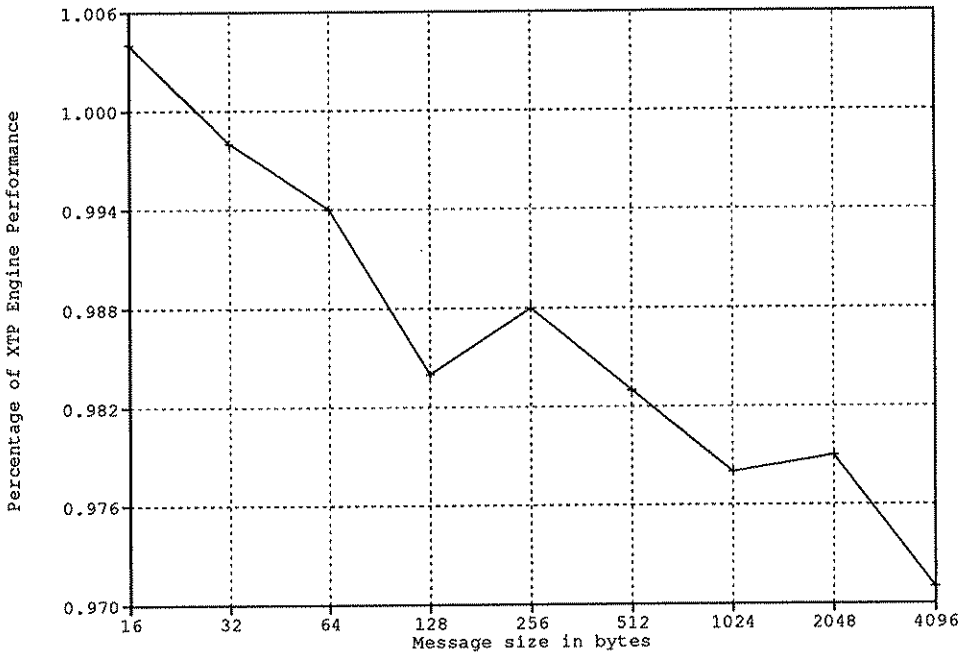


**Figure 3.4** — *XTP Driver Round-trip Latency Efficiency on Token Ring*

### 3.2.2. IEEE 802.3 Ethernet Implementation

#### 3.2.2.1. Throughput Measurements

In Figure 3.5 we see a plot of message size on the horizontal axis versus throughput on the vertical axis. The 802.3 Ethernet version of XTP memory transfer driver provides just under 4.7 Mbits/sec of throughput when using 8 Kbyte messages. The Ethernet version of XTP, unlike the Token Ring version, does not provide 100% efficiency between the XTP memory transfer driver and the low level interface; however, as we will see in Figure 3.2 it is still extremely efficient.

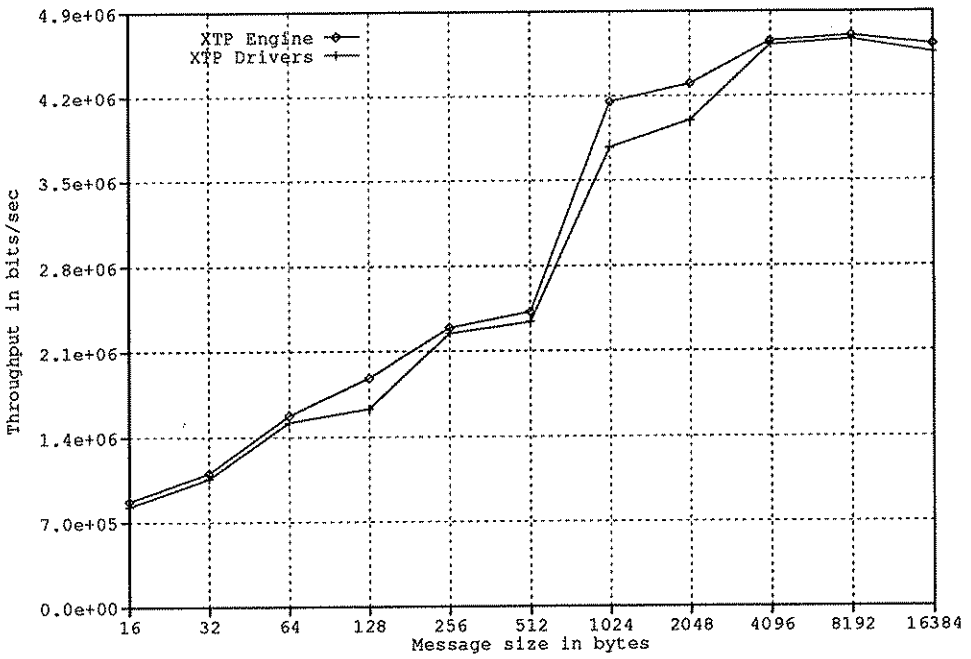


**Figure 3.5** — *XTP Unicast Throughput on Ethernet*

In Figure 3.2 we see a graph with message size plotted on the horizontal axis and the XTP memory transfer driver efficiency plotted on the vertical axis. Efficiency is

expressed in terms of the percentage of the XTP low level interface performance delivered by the memory transfer driver. XTP memory transfer driver delivered an average of 95% of the throughput performance of the low level interface using the 802.3 Ethernet MAC processor. While this is not perfect efficiency, it falls within the targeted design parameters for the XTP driver interfaces [12].

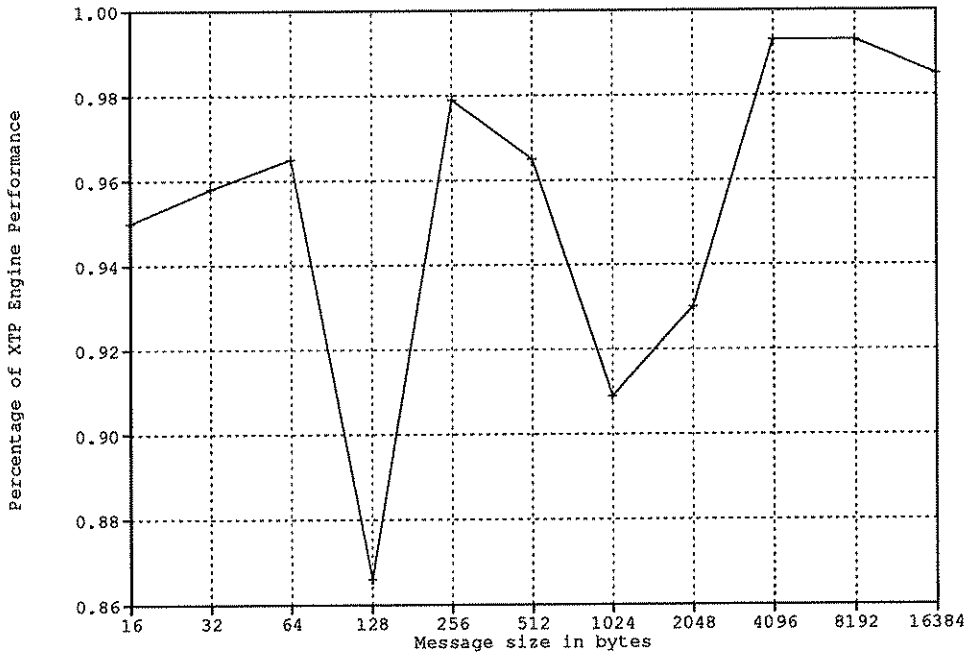


**Figure 3.6** — *XTP Driver Throughput Efficiency on Ethernet*

As a further indication of the efficiency of both the XTP driver interfaces and the XTP Engine, we have measured the maximum throughput of the WD8003E MAC processor to be 5.5 Mbits/sec. This means that XTP is delivering over 85% of the throughput performance of the MAC interface in a reliable data transfer.

### 3.2.2.2. Round-trip Latency Measurements

In Figure 3.7 message size is plotted on the horizontal axis versus *round-trip* latency on the vertical axis. The IEEE 802.3 version of XTP using the memory transfer driver provides a *round-trip* latency of 2.31 ms for a message of 16 bytes. As with the *round-trip* latency measurements for Token Ring implementation of XTP, the *round-trip* latency curve of the Ethernet implementation is also nearly linear with message size.

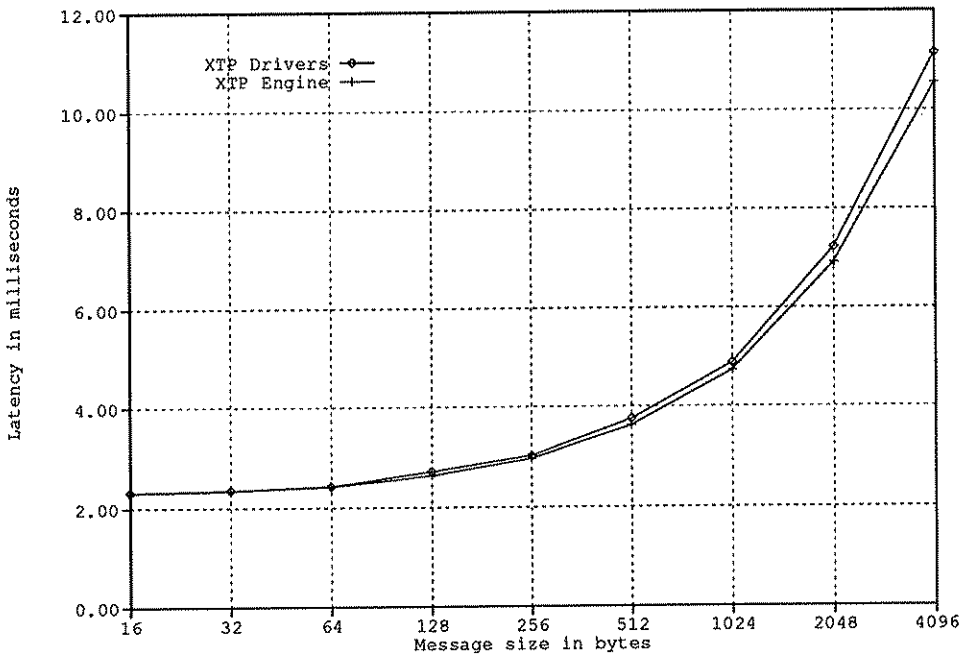


**Figure 3.7** — *XTP Unicast Round-trip Latency on Ethernet*

Figure 3.8 plots message size on the horizontal axis versus *round-trip* latency efficiency on the vertical axis. Efficiency is expressed in terms of the percentage of XTP low level interface *round-trip* latency performance delivered by the XTP memory transfer driver. For a 16 byte message the memory transfer driver has *round-trip* latency

performance only 0.4% worse than the *round-trip* latency performance offered by the low level interface performance. Averaging across all message sizes, the XTP memory transfer driver delivers *round-trip* latency performance which is approximately 2% worse than the *round-trip* latency performance offered by the XTP low level interface.

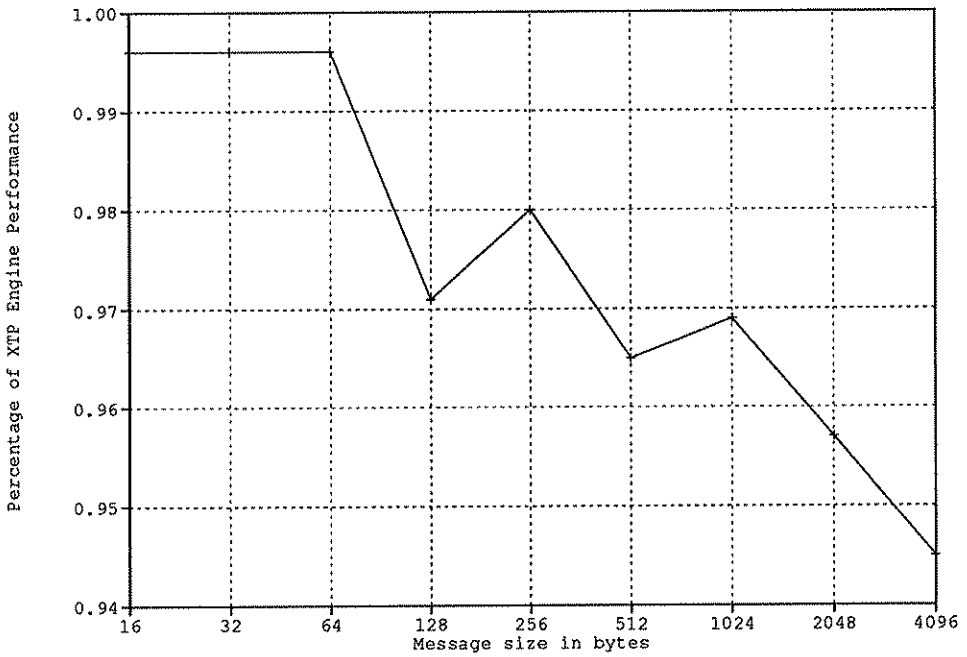


**Figure 3.8** — *XTP Driver Round-trip Latency Efficiency on Ethernet*

### 3.3. XTP Multicast Measurements

The best-effort multicast facility provided by XTP is a truly unique feature not offered by any other transport protocol. Unlike VMTP's multicast facility, XTP uses the bucket algorithm to allow the transmitter to accumulate control information about the receiver set. The use of slotting and damping to reduce *Cntl* packet traffic is also unique to XTP. In the sections which follow we will present results of performance experiments which indicate that XTP's multicast facility is extremely efficient in providing high throughput group communications. We will also show some performance numbers which indicate the effectiveness of the bucket algorithm and the slotting and damping procedure.

#### 3.3.1. Multicast Groups

For our multicast experiments we used all seven machines at our disposal in the UVa Computer Networks Lab in order to perform experiments for multicast groups with two to six receivers. The transmitter for all multicast experiments was a class 4 machine. When moving from a 1-to-n multicast group experiment to a 1-to-n+1 multicast group experiment the fastest machine available was always added to the group. For example, our 1-to-2 multicast group had a class 4 machine as transmitter and two class 1 machines as receivers. Our 1-to-3 multicast group had a class 4 machine as transmitter and two class 1 and one class 2 machine as receivers. Included in all graphs of multicast throughput performance is a curve which represents unicast throughput performance. This unicast throughput performance curve represents the unicast throughput obtained by using a class 4 machine as transmitter and a class 1 machine as receiver.

### 3.3.2. IEEE 802.5 Token Ring Implementation

#### 3.3.2.1. Throughput Measurements

In Figure 3.9 we see a graph with message size on the horizontal axis and throughput on the vertical axis. Each curve represents the throughput as it appears to the transmitter using the XTP memory transfer driver. A throughput of approximately 985 Kbits/sec is achieved in a 1-to-4 multicast using 4 Kbyte messages. The unicast throughput using 4 Kbyte messages is 991 Kbits/sec. The multicast facility in this case provides reliable data transfer service to four stations 4 times faster than the unicast facility could provide the same service to the same four stations.

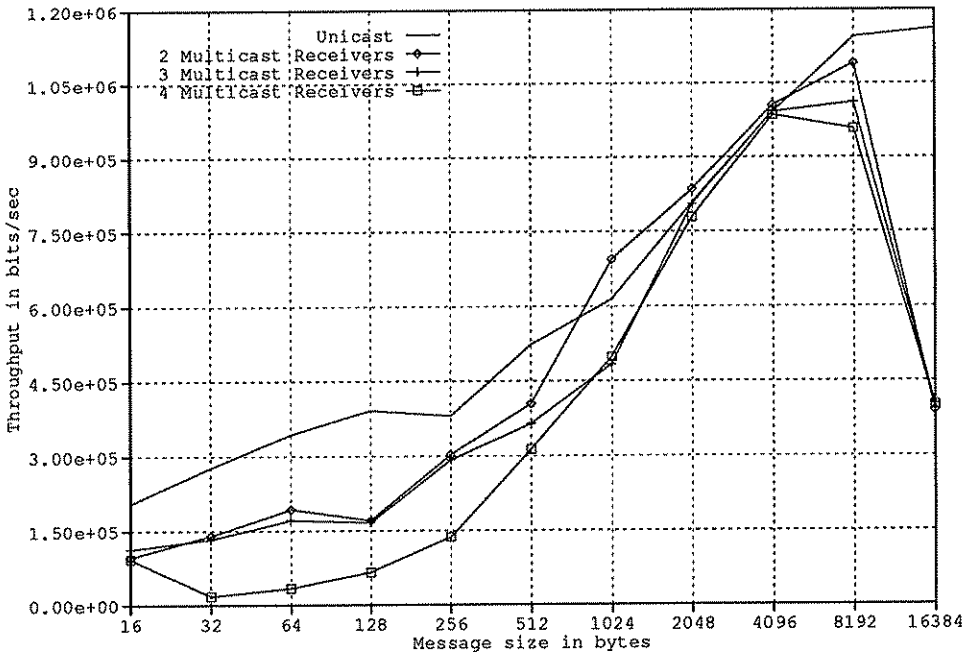


**Figure 3.9** — *XTP Multicast Throughput on Token Ring*

Figure 3.9 also has an anomaly in the fact that the 1-to-2 multicast throughput actually exceeds the unicast throughput for message sizes between 1 Kbyte and 4 Kbytes. This phenomenon is the result of the delay introduced at the transmitter by the bucket algorithm. The overhead of the bucket algorithm paces the transmitter so that the receivers and transmitter are more closely matched in terms of performance. Because the transmitter and the receivers are more closely matched, fewer packets are dropped by the receivers and throughput improves.

### 3.3.3. IEEE 802.3 Ethernet Implementation

#### 3.3.3.1. Throughput Measurements

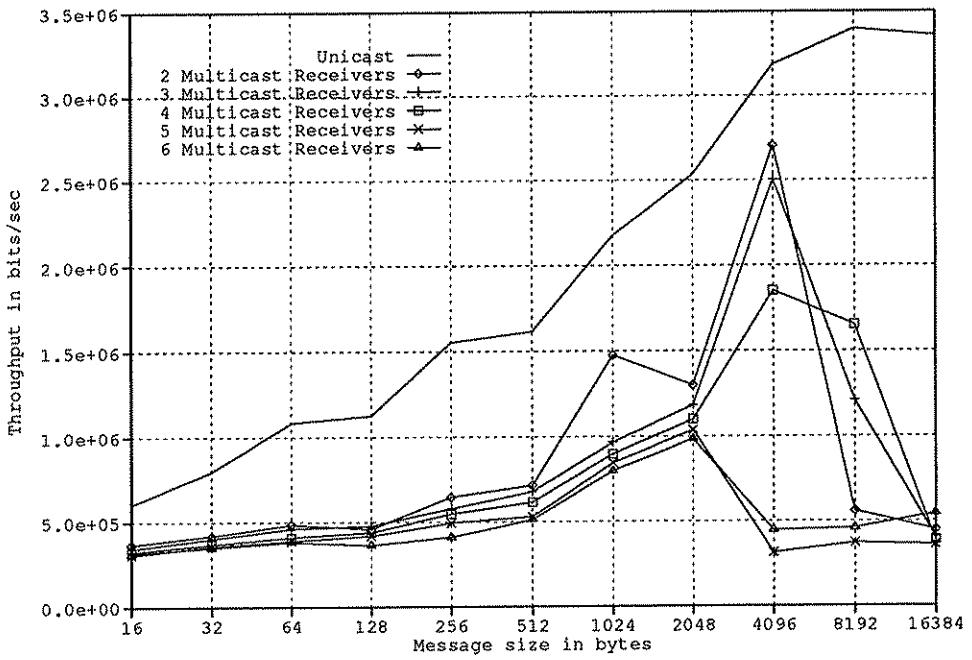


**Figure 3.10** — *XTP Multicast Throughput on Ethernet*

In Figure 3.10 we see a graph with message size plotted on the horizontal axis and throughput plotted on the vertical axis. Each curve represents the throughput attainable by the transmitter using the XTP memory transfer driver.

For a 1-to-6 multicast the transmitter would see throughput of 985 Kbits/sec using a message of 2 Kbytes. The unicast throughput using 2 Kbyte messages is 2.53 Mbits/sec. In order to unicast the data to six receivers as quickly as the multicast facility, a unicast throughput of 5.91 Mbits/sec would be required. For this particular case, XTP's multicast facility provides an effective throughput more than two times faster than that of unicast.

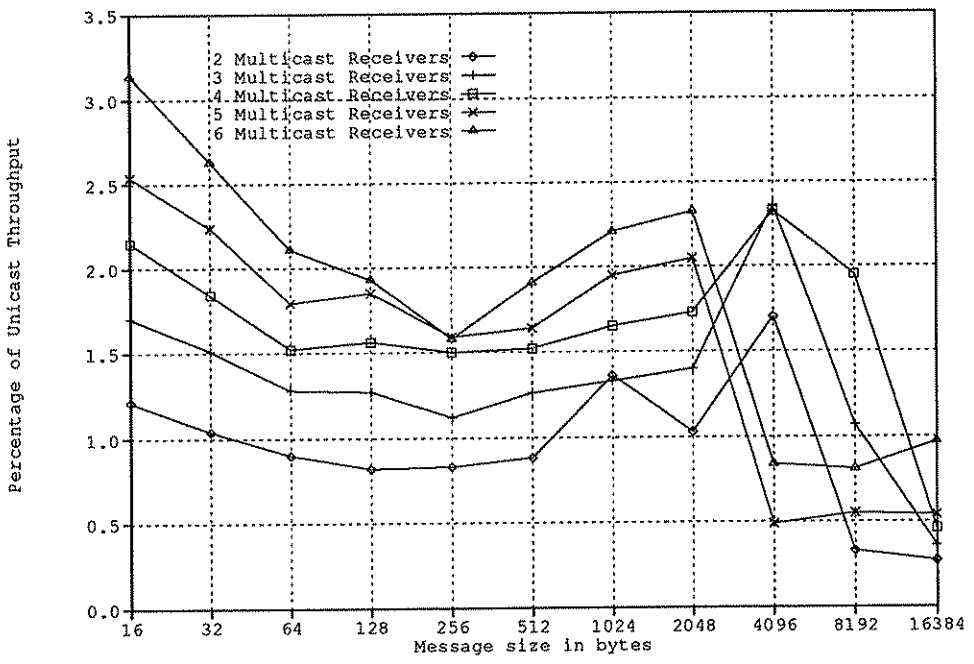


**Figure 3.11** — *XTP Multicast Throughput Efficiency on Ethernet*

Figure 3.11 plots message size on the horizontal axis and efficiency on the vertical axis. In this case efficiency is measured as the percentage of the XTP unicast throughput

delivered by the XTP multicast facility. For a 1-to-6 multicast group using 1 Kbyte messages the XTP multicast facility delivers 221% of the XTP unicast throughput.

In analyzing Figure 3.11 it is clear that for most cases XTP's multicast facility provides a large performance gain over unicast. The exceptional cases include the 1-to-2 multicast group and some cases involving message sizes of 4, 8 and 16 Kbytes. In the next section dealing with the bucket algorithm we will discuss the cause of the loss of efficiency for cases which involve large message sizes.

#### 3.3.3.2. The Bucket Algorithm

The Bucket Algorithm described in the XTP 3.5 specification uses the expiration of a timer called the STIMER (switch timer) to demarcate the end of one bucket and the beginning of the next [17]. When the STIMER expires the current bucket is cycled to the end of the bucket list and the contents of the bucket at the front of the bucket list are applied to the state of the protocol engine. In addition, a SREQ *Cntl* packet is sent to the multicast group. The bucket from the front of the bucket list becomes the current bucket and all responses to the most recently sent SREQ *Ctnl* packet are accumulated in that bucket. The amount of time used to initialize the STIMER is dependent on the reliability desired and the number of buckets.

Unlike the Bucket Algorithm described in the XTP 3.5 specification, the Bucket Algorithm implemented in UVa XTP 3.5 is based on messages, not on timers. Each message presented to the XTP engine causes a bucket cycle to occur. The algorithm was implemented this way because the IBM PC/AT does not have enough high resolution timers to handle all the timing functions required by the XTP transmitter. In fact the IBM PC/AT does not have any interrupt driven clocks with enough resolution to handle the requirements of the STIMER.

The variation of the Bucket Algorithm implemented in UVa XTP 3.5 causes some interesting interactions between the message size, the data ring size, the number of buckets, the WTIMER (wait timer) and the multicast throughput. The WTIMER is a coarse grained timer (55 ms) set when an SREQ *Cntl* packet is transmitted. If a response is not received from one of the multicast receivers within the period of the WTIMER another SREQ *Cntl* packet is sent to the multicast group. As a result each bucket has a maximum period of one WTIMER period or 55 ms.

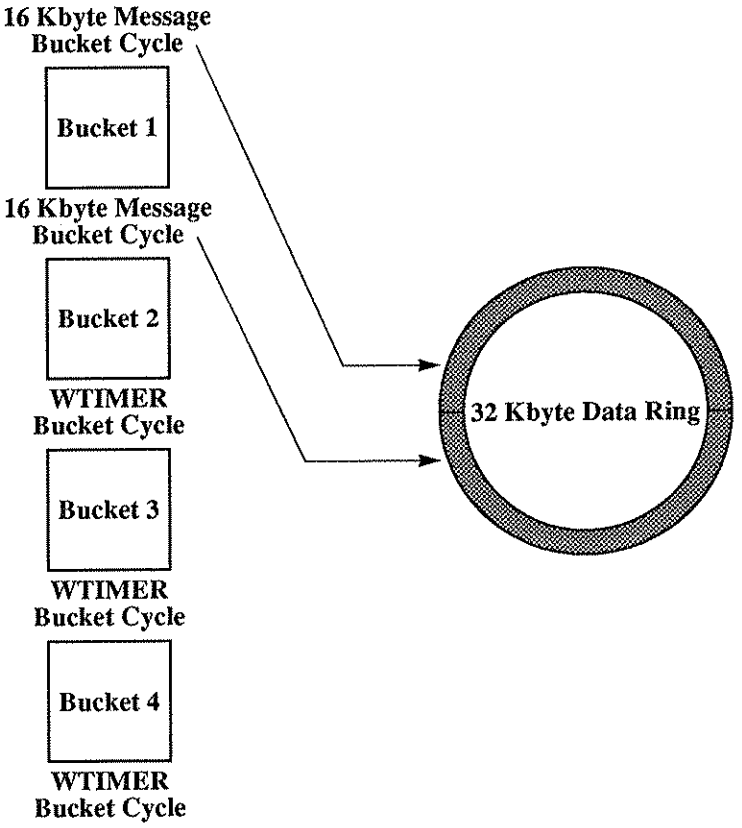


**Figure 3.12** — *UVa XTP 3.5 Bucket Algorithm*

The problem with using messages as bucket delimiters is illustrated in Figure 3.12 using a system with four buckets and a 32 Kbyte data ring. In the diagram the first 16 Kbyte message fills half the data ring and begins the period for bucket 1. The second 16 Kbyte message fills the other half of the data ring and begins the period of bucket 2. Because there is no more space for messages in the data ring the XTP memory transfer driver blocks, waiting for ring space to be freed. Ring space cannot be cleared until bucket 1 cycles back to the front of the list and applies its contents to the state of the protocol engine. Unfortunately, bucket 1 cannot cycle back to the front of the bucket list until the bucket periods of buckets 2, 3, and 4 have expired. This requires three expirations of the WTIMER or 165 ms.

There are two possible solutions for the problem described above. The first is to decrease the number of buckets in order to decrease the period of deadlock. This solution will work as long as all the receivers can keep up. If one or more of the receivers is falling behind, fewer buckets will cause the slower receivers to fall out of the group.

Figure 3.13 illustrates quite clearly that this solution will work as long as all the receivers are of the same general class, as is the case with our 1-to-4 multicast group. Plotted on the horizontal axis is message size and plotted on the vertical axis is throughput. As the number of buckets is decreased from 8 to 4, and then from 4 to 2, the throughput for the larger message sizes increases dramatically. Sixteen Kbyte messages will always present a problem because they will still require one WTIMER expiration even when using only two buckets.

The other possible solution is to increase the data ring size to 64 Kbytes. Unfortunately, because this implementation of UVa XTP 3.5 is based on the Intel 80x86 architecture this is a slightly tricky proposition. It is possible but requires some major modifications to the UVa XTP protocol engine.

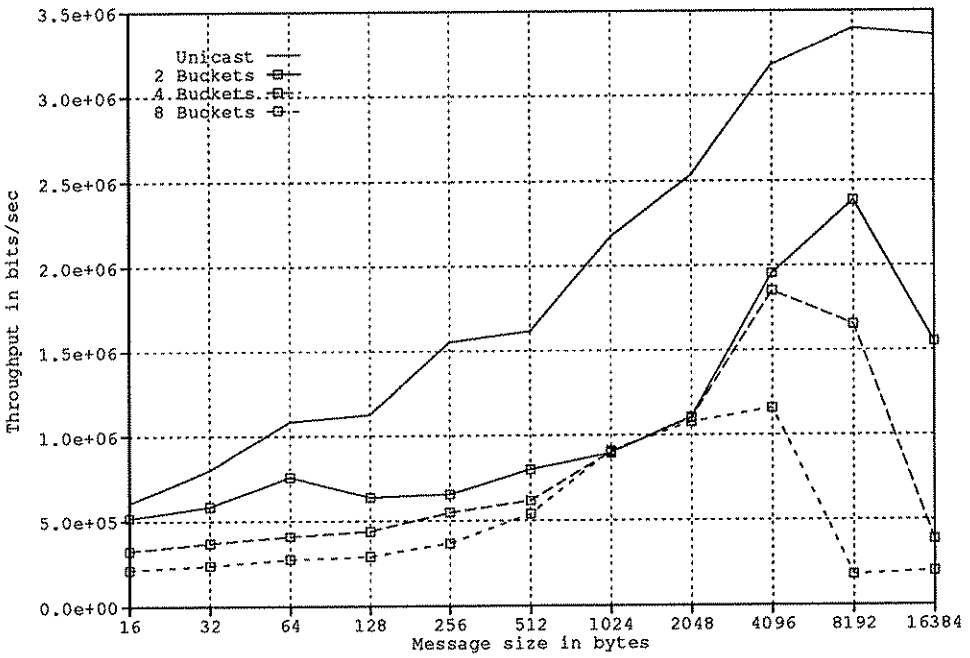


**Figure 3.13** — *XTP Four Receiver Multicast Throughput*

### 3.3.3.3. Slotting And Damping

The XTP 3.5 protocol specification describes the slotting and damping procedure as highly implementation dependent [17]. In particular, the size and number of time slots, as well as the method used for time slot assignment, is left to the protocol implementor. The following paragraphs will describe how the slotting and damping procedure is implemented in UVa XTP 3.5.

Each slot in XTP is equivalent to 32 ticks of the IBM PC/AT system clock, or approximately 27 μs. Slots are assigned by using a pseudo random number generator to

provide a random integer in the range 0 to 500. The random number generator is seeded with an integer derived from the local station's MAC address.

The damping procedure is implemented in UVa XTP 3.5 in the following manner. Each context structure has a set of control bits associated with it. When a SREQ *Cntl* packet arrives at a local context the SREQ bit is set in the context structure. The next time the local context receives service from the context processor the SREQ bit will be checked. If it is still on, a *Cntl* packet will be sent with the appropriate control information. Alternatively, if a *Cntl* packet from another receiver in the same multicast group arrives before the local context is serviced, the control values of that *Cntl* packet will be compared with the local values. If the local values are redundant the local context's SREQ bit will be masked. In this way, the context processor is prevented from sending a *Cntl* packet.

There are two basic metrics of interest with respect to the slotting and damping procedure — *Cntl* packet traffic and throughput. Specifically, how much does slotting and damping reduce multicast receiver-generated *Cntl* packet traffic, and how does the use of slotting and damping affect the throughput performance of the XTP multicast facility?

In Figure 3.14 we see the effects of slotting and damping on receiver-generated *Cntl* packet traffic for a 1-to-4 multicast group. Plotted on the horizontal axis is the message size. Plotted on the vertical axis is the total number of *Cntl* packets transmitted by the receivers of the multicast group during the reception of 2 Mbytes of data. Each curve represents one alternative for the activation of the slotting and damping procedures.

As we can see, when slotting and damping are both active the number of *Cntl* packets transmitted by the receivers is significantly less than for the other three alternatives. Averaged over all message sizes, when slotting and damping are both enabled there are 21% fewer receiver-generated *Cntl* packets transmitted than when slotting and damping are

both disabled. This is outstanding given that the slotting and damping procedure was designed and intended to be most effective for multicast groups of 1-to-10 or larger [17].

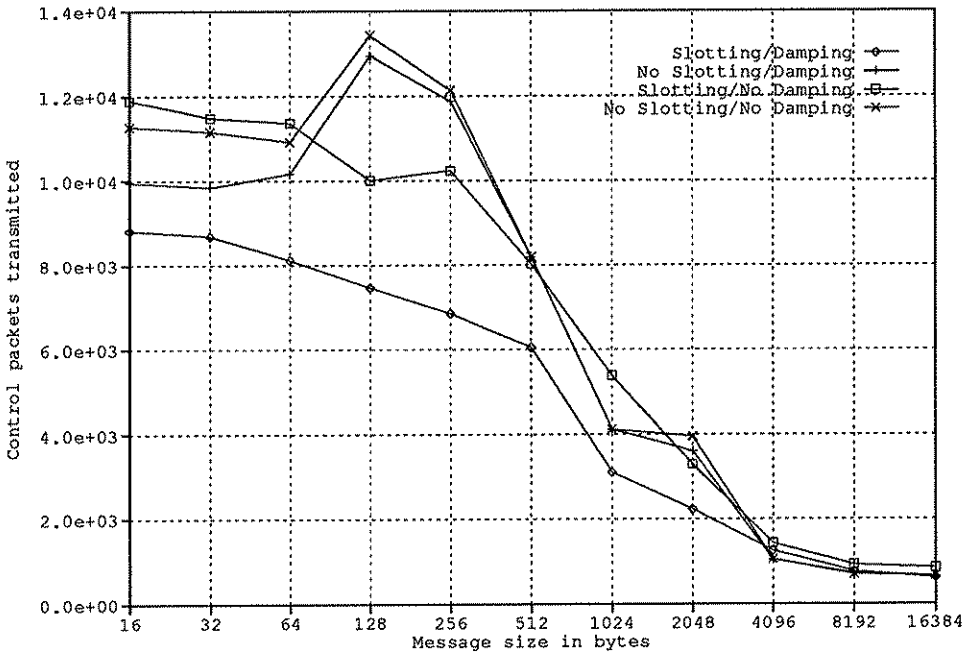


**Figure 3.14**—*XTP Multicast Slotting and Damping Control Packet Transmission*

Figure 3.15 shows the effect that slotting and damping have on multicast throughput performance. Message size is plotted on the horizontal axis versus throughput on the vertical axis. From the graph it is clear that slotting has a negative effect on throughput performance. Since slotting causes receivers to delay responding to SREQ *Cntl* packets, the transmitter's data ring space is freed more slowly and the XTP memory transfer driver will tend to block, waiting for service.

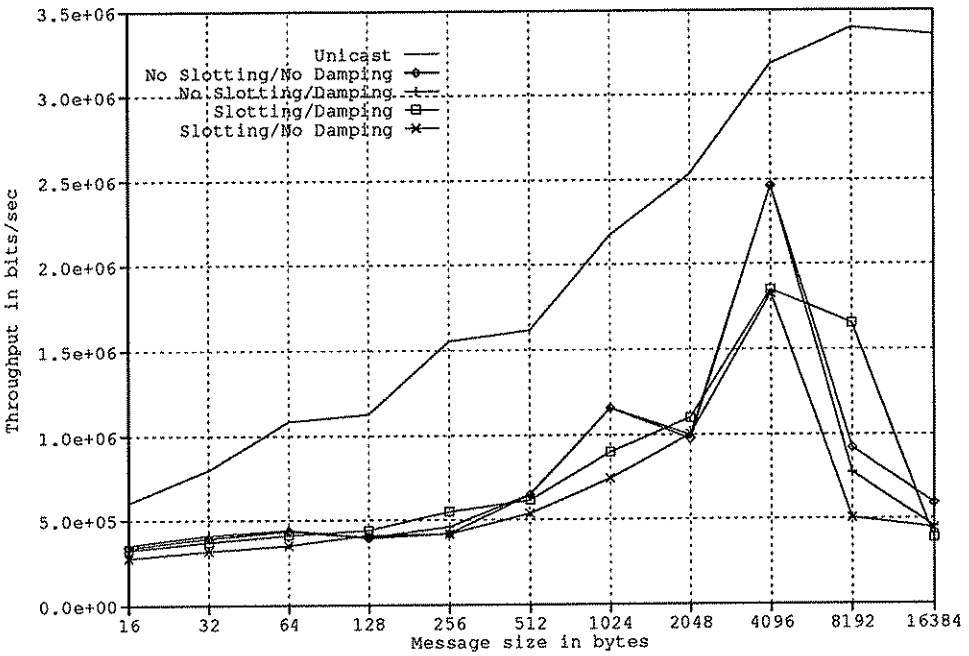


**Figure 3.15** — *XTP Multicast Slotting and Damping Throughput*

### 3.4. XTP vs. TCP/IP Measurements

In order to reach some conclusions about the performance of the UVa Computer Networks Lab's implementation of XTP 3.5 relative to other transport protocols, we tested the performance of an implementation of TCP/IP running on the identical hardware platform. Of course, the important part of these experiments was determining the circumstances under which comparisons between UVa XTP 3.5 and TCP/IP were valid. We chose unicast throughput, *one-way* latency and resource allocation delay measurements as the three areas of valid comparison.

All the experiments were performed using a class 2 machine (see Table 2.1) as both transmitter and receiver. In addition, because it is not possible to disable the TCP/IP checksum algorithm, we were required to enable the UVa XTP 3.5 data checksum algorithm in order to make our comparisons valid. It should be noted, however, that the checksum algorithm used in TCP is a simple algorithm based on addition of one's-complement 16-bit words [15]. The checksum algorithm used in XTP is a 32-bit checksum based on a series of exclusive ORs and bitwise rotations [17]. The XTP checksum algorithm is much more computationally expensive than the TCP checksum algorithm.

#### 3.4.1. WIN/TCP for DOS

The implementation of TCP/IP chosen for our experiments was WIN/TCP for DOS 4.0 from The Wollongong Group Inc. [31]. WIN/TCP for DOS 4.0 is an IBM PC/AT hosted implementation of TCP/IP based on an implementation called PC/IP originally developed at MIT. The WIN/TCP kernel is implemented as a DOS TSR (Terminate Stay Resident) program which occupies PC memory above 640K. Included with the implementation are device drivers for a variety of LAN interfaces including the Western Digital WD8003E Ethernet interface used with the UVa Computer Networks Lab's implementation of XTP 3.5.

### 3.4.2. WIN/API for DOS

All access to the memory-resident WIN/TCP kernel is implemented via a set of interrupt service routines. The exact nature of the interrupt service routine interface to the kernel is proprietary. Therefore, a high level application programmer's interface is required for application programs to operate in conjunction with the WIN/TCP kernel. For this purpose The Wollongong Group provides WIN/API for DOS 4.0 [32].

WIN/API for DOS 4.0 provides two sets of interface routines. One set of routines provides a low level interface to the WIN/TCP kernel. The low level interface requires the application programmer to provide upcall routines to handle various events which may occur in the WIN/TCP kernel. This low level interface is fairly complicated and cumbersome to work with. The other set of routines provides an interface virtually identical in name and function to the socket IPC interface provided by BSD UNIX 4.3. The two sets of interface routines are packaged in four MicroSoft C 5.1 compatible libraries which can be linked with application object files using the MicroSoft linker. All application programs used to test WIN/TCP for DOS were developed using the WIN/API socket IPC library.

### 3.4.3. Throughput Measurements

In Figure 3.16 we see a graph with message size plotted on the horizontal axis and throughput plotted on the vertical axis. As we can see, UVa XTP 3.5 with the data checksum algorithm disabled provides better throughput performance than TCP/IP for all message sizes. XTP with the data checksum enabled has better throughput performance than TCP/IP for all message sizes except 256 and 512 bytes.

The maximum throughput delivered by TCP/IP is 2.68 Mbits/sec using messages of 4 Kbytes. The throughput delivered by XTP with the data checksum enabled using 4 Kbyte messages is 3.16 Mbits/sec. That is 18% better throughput performance than TCP/IP using

4 Kbyte messages. XTP with the data checksum enabled attains a maximum throughput of 3.18 Mbits/sec using 8 Kbyte messages. That is a 33% improvement over TCP/IP using 8 Kbyte messages.

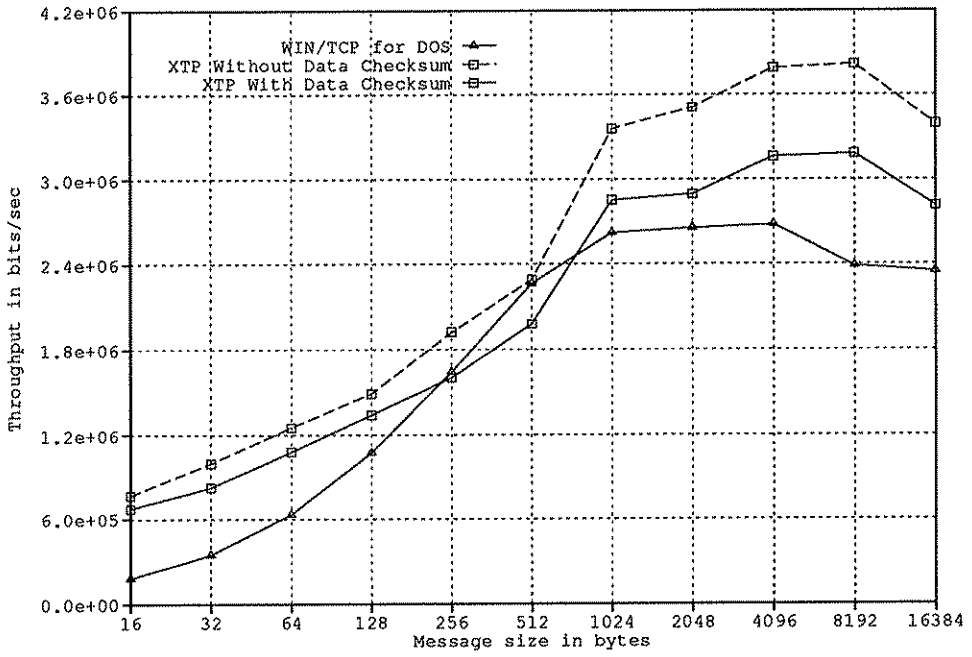


**Figure 3.16** — *TCP vs. XTP Throughput*

### 3.4.4. One-way Latency Measurements

In Figure 3.16 we see a graph with message size plotted on the horizontal axis and *one-way* latency plotted on the vertical axis. TCP/IP has a *one-way* latency of 1.09 ms when using a 16 byte message. XTP with the data checksum enabled has a *one-way* latency of 2.03 ms. The XTP and TCP curves are nearly linear with message size.

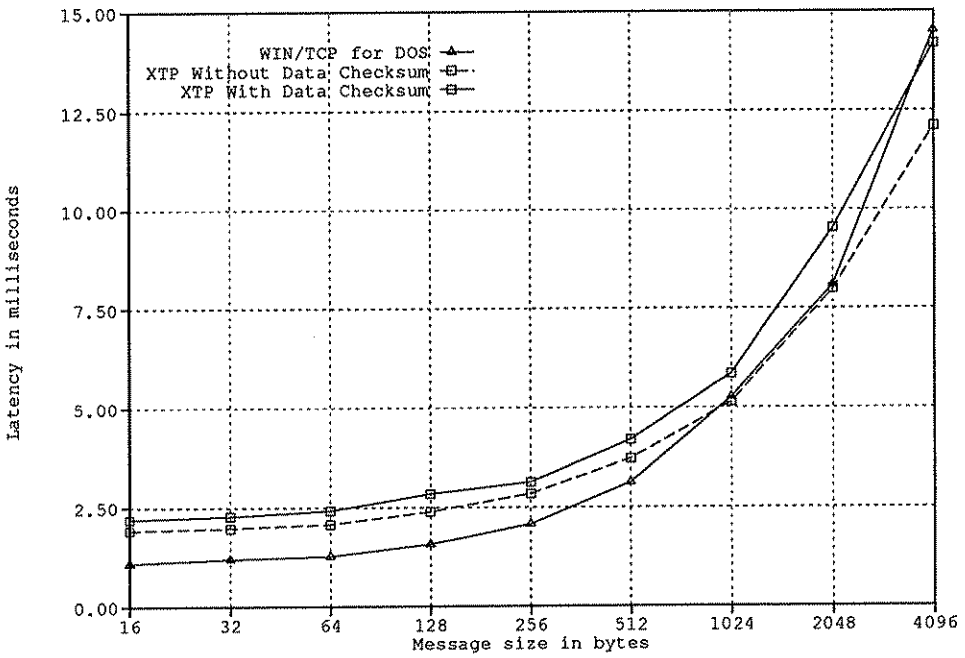


**Figure 3.17** — *TCP vs. XTP One-way Latency*

UVa XTP 3.5's *one-way* latency performance is rather poor compared to TCP/IP. The reason for this is the transmission syntax used by the XTP memory transfer driver. It requires the transmission of four packets (2 *Data*, 2 *Cntl*) to send a 16 byte message to a destination and then return it. TCP/IP only requires two packet transmissions to perform the same operation. This is not an inherent problem with the XTP protocol, but rather a feature of the protocol syntax implemented by the XTP memory transfer driver.

### 3.4.5. Delay Measurements

The delay measurement of interest is the total time required to open a connection and send a message and then close the connection. In Figure 3.18 we see a graph with

message size plotted on the horizontal axis and delay plotted on the vertical axis. As can be seen in Figure 3.18, the speed with which XTP can set up a connection, send a message and close the connection far exceeds TCP's performance when providing the same service. XTP with the data checksum enabled requires 4.11 ms to establish a connection, send a 16 byte message and close the connection. It requires 17.32 ms for TCP to perform the same operations. Because the XTP *First* packet can carry data, opening the connection and sending the message are achieved simultaneously. TCP requires a fixed overhead of 2.42 ms to open a connection and approximately 14.20 ms to close the connection. XTP requires 3.41 ms to close a connection. Both sets of curves are nearly linear with message size.

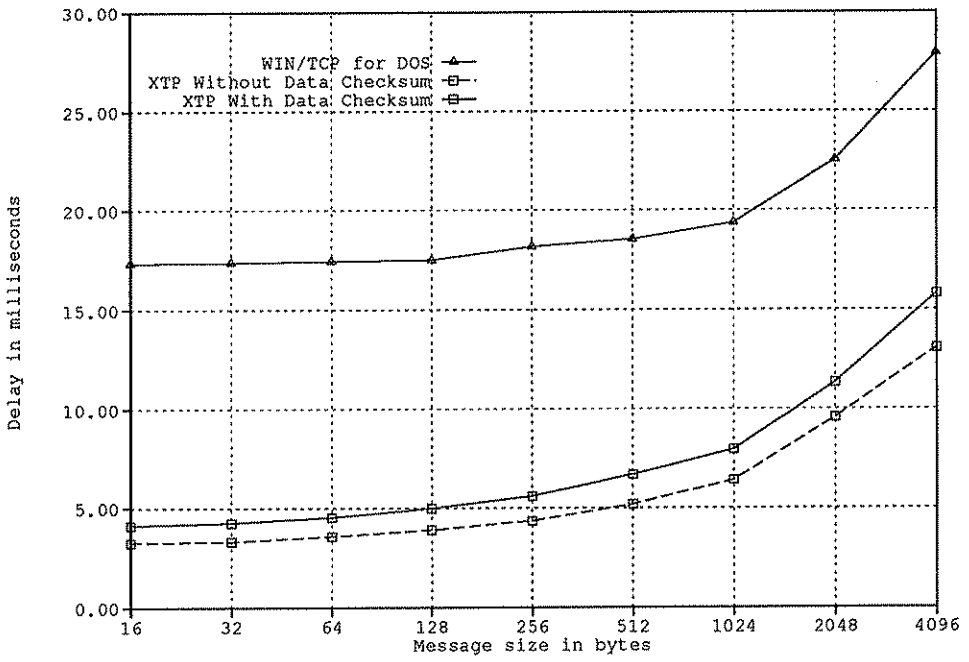


**Figure 3.18** — *TCP vs. XTP Connection Setup Delays*

*Chapter 4*

# Conclusions

## 4.1. Transport Protocol Survey

In our transport protocol survey we discussed a wide range of transport protocols and transport protocol functionality. We examined TCP and TP4, two traditional connection oriented transport protocols. We discussed VMTP, a transport protocol designed for distributed systems applications based on request/response transmission semantics. The GAM-T-103 architecture specification for military real-time local area networks was examined in order to conclude what networking functionality is required for embedded real-time systems. Lastly, we examined the Xpress Transfer Protocol. XTP implements many of the features of the traditional connection oriented protocols and VMTP, and also offers functionality not supported by TCP, TP4 or VMTP. In the sections which follow we will examine some general features of transport protocols in order to conclude where XTP exceeds the capabilities of the other transport protocols we discussed.

### 4.1.1. Data Communication Syntaxes

TCP and TP4 essentially offer one type of data communication syntax — a duplex connection-oriented syntax. They require a connection to be explicitly established and explicitly closed. In the case of TP4, it requires six packet transmissions to send 1 byte of user data. Both protocols have complex packet headers which can have variable size.

VMTP has essentially one data communication syntax as well. It uses a request/ response communication syntax. It has only two fixed size packet types. VMTP is very well suited to client/server distributed systems applications. However, VMTP does not support streaming very well and it does not have any support for unacknowledged datagrams.

XTP has a wide range of data communication syntaxes. It can support a traditional connection-oriented byte stream syntax, a request/response message syntax, an acknowledged datagram syntax and an unacknowledged datagram syntax. The reason that XTP has this wide range of function is the result of two features of the protocol. One is the ability of the XTP *First* packet to carry user data. Another is the master/slave relationship between the XTP transmitter and receiver. The XTP receiver takes no independent action with respect to acknowledging data. It only acts in response to commands from the transmitter. This allows the XTP transmitter to implement a variety of communication syntaxes by varying the way in which commands are passed to the receiver.

### 4.1.2. Sequencing and Error Control Mechanisms

TCP and TP4 both use a sliding window and variations on the go-back-n retransmission strategy to implement sequencing and error control. These schemes are acceptable for use with high error rate networks which cause sequences of packets to be destroyed. However, on newer technologies such as FDDI, packet overrun is the major cause of lost packets and therefore a selective retransmission scheme would be much more efficient in filling the occasional gap in the data stream.

Because VMTP is not a stream-based protocol it has no use for a sliding window mechanism. It does, however, have a selective retransmission scheme which is implemented on a per-message basis. Each message control block has a bitmask which allows blocks of the message to be independently acknowledged. In this way only the missing blocks need be retransmitted.

XTP implements a sliding window and also implements a selective retransmission scheme based on the positive acknowledgment of the correctly received spans of data. By

manipulating these spans correctly XTP can also implement a go-back-n retransmission strategy.

### 4.1.3. Flow Control

Flow control is a mechanism used to manage receiver buffer space efficiently. It is usually implemented using a buffer reservation scheme. TCP, TP4 and XTP implement a buffer reservation flow control mechanism. These protocols allow the receiver to inform the transmitter of the amount of buffer space available at the receiver. The transmitter cannot exceed that buffer allocation until further buffer credit is allocated by the receiver. Alternatively, because VMTP is not stream-based it does not implement any form of flow control for buffer management.

### 4.1.4. Rate and Burst Control

Rate control is a mechanism used to combat the problem of packet overrun. It is designed to provide an inter-packet gap long enough to allow the receiver to keep up. Similarly, burst control is designed to limit the amount of data which can be contained in any one burst of packets. Because TCP and TP4 were designed during a period when slow networks were the norm and packet overrun was not a problem, neither protocol has any provision for rate and burst control.

VMTP has a rate control mechanism based on the 8-bit inter-packet gap field in the packet header. Responses and acknowledgments sent by the receiver communicate to the transmitter the inter-packet gap which is acceptable to the receiver. VMTP burst control is implemented implicitly by the protocol's limitation on message size.

XTP implements rate and burst control by using the *rate* and *burst* fields contained in the XTP *Cntl* packet. The XTP *rate* field corresponds to the maximum number of octets to be transmitted per second. The *burst* field indicates the amount of data to be transmitted

per burst of packets. Using these fields the receiver can effectively throttle the transmitter. In addition, because of XTP's transfer layer architecture, XTP routers can participate directly in the negotiation of the *rate* and *burst* parameters. This is a very important feature because in many cases routers are the bottleneck in an internetwork.

### 4.1.5. Multicast Transmission

With respect to multicast, only VMTP and XTP support this functionality. Both protocols implement a best-effort multicast facility; however, XTP is different from VMTP in two respects. First, XTP can efficiently implement a stream-based multicast communication. VMTP's multicast facility uses request/response semantics. Second, with the use of the bucket algorithm XTP can utilize responses from more than one of the multicast receivers when updating the state of the multicast transmitter. VMTP multicast transmitters disregard any multicast receiver response which arrives after the first multicast receiver response.

### 4.1.6. Hardware Support and Implementation

TCP and TP4 were clearly designed for software implementaion and VMTP offers the ability to implement hardware support for checksum calculation. XTP, however, was designed specifically to be implemented using VLSI technology. XTP's fixed header and trailer format coupled with its transfer layer architecture make it uniquely suited for complete hardware implementation.

## 4.2. Performance Analysis

In our performance analysis of UVa XTP 3.5 we performed a wide range of experiments on XTP in order to draw some conclusions about the performance capabilities of the implementation. We presented the results of throughput and latency measurements of XTP's unicast facility and the results of throughput measurements of XTP's multicast

facility. We also presented some experimental results which shed some light on the effectiveness of the bucket algorithm and the slotting and damping procedure. Lastly, we presented a set of performance measurements comparing the throughput, latency and delay characteristics of UVa XTP 3.5 with the throughput, latency and delay characteristics of an implementation of TCP. In the sections which follow we will summarize the results of our experiments.

### 4.2.1. XTP Unicast Performance

#### 4.2.1.1. Throughput Performance

The unicast throughput performance of the IEEE 802.5 Token Ring version of UVa XTP 3.5 was very impressive. The throughput performance measurements indicated that the MAC processor was definitely the bottleneck in the system. The token ring implementation achieved a maximum throughput of 2.21 Mbits/sec on a 4 Mbits/sec Token Ring.

The IEEE 802.3 Ethernet implementation also had very impressive throughput performance. UVa XTP 3.5 achieved a reliable throughput of 4.7 Mbits/sec using a MAC processor that had a maximum throughput of 5.5 Mbits/sec. XTP delivered over 85% of the bandwidth of the MAC processor in a reliable data transfer.

#### 4.2.1.2. Round-trip Latency Performance

The *round-trip* latency performance of the IEEE 802.5 Token Ring implementation of UVa XTP 3.5 was extremely efficient. The XTP memory transfer driver only degraded the latency performance of the XTP engine by an average of 1%.

The *round-trip* latency performance of the IEEE 802.3 Ethernet version of UVa XTP 3.5 was also extremely efficient. There was only a 2% degradation in performance when using the XTP memory transfer driver instead of the XTP low level engine interface.

### 4.2.2. XTP Multicast Performance

#### 4.2.2.1. Throughput Performance

Our throughput measurements of XTP's multicast facility indicated that in some cases it provided reliable throughput performance more than 3 times faster than the performance of the XTP unicast facility. Our experiments also indicated that there was a complex relationship between the throughput performance of the XTP multicast facility, the message size, the data ring size and the number of buckets utilized by the bucket algorithm.

#### 4.2.2.2. Bucket Algorithm Performance

The performance of the version of the bucket algorithm implemented in UVa XTP 3.5 was mixed. It performed its function in that it allowed the multicast communication to be tuned relative to the desired reliability and performance. However, it interacts in an undesirable fashion with the message size and the size of the data ring. The problems with the bucket algorithm have been recognized and are being corrected.

#### 4.2.2.3. Slotting and Damping Performance

The performance of the slotting and damping procedure was much better than expected. The slotting and damping procedure was intended to be most effective when used with large multicast groups containing many homogeneous machines. Our multicast groups were neither large nor homogeneous and yet we observed as much as a 44% reduction in the receiver generated *Cntl* packet traffic in a 1-to-4 multicast group.

### 4.2.3. XTP vs. TCP Performance

#### 4.2.3.1. Throughput Performance

Our performance measurements of UVa XTP 3.5 and TCP indicated that XTP provided as much as 33% better throughput performance than TCP. XTP achieved a

maximum throughput of 3.18 Mbits/sec using a message size of 8 Kbytes. TCP achieved a maximum of 2.39 Mbits/sec using the same message size. In addition, a significant performance penalty was paid by XTP because the checksum algorithm employed by XTP is much more complicated than the checksum algorithm implemented by TCP.

#### 4.2.3.2. One-way Latency Performance

The *one-way* latency performance of XTP was not impressive when compared with TCP. The *one-way* latency of a 16 byte message on UVa XTP 3.5 was 2.20 ms. The TCP *one-way* latency for the same size message was 1.09 ms. The chief reason for the poor performance of XTP relative to TCP is related to the communication syntax used by the XTP memory transfer driver. XTP required four packet transmissions to send the message and return it. TCP required only two packet transmissions to perform the same operation.

#### 4.2.3.3. Delay Performance

Our delay measurements indicated that UVa XTP 3.5 has a large performance advantage over TCP with respect to the opening and closing of connections. It required over 17 ms for TCP to open a connection, send a 16 byte message, and close the connection. It required just over 4 ms for XTP to perform the same set of operations. These measurements clearly illustrate the utility of the XTP *First* packet for providing a high performance acknowledged datagram service.

### 4.3. Conclusion

When looking at the full range of features offered by the protocols we examined in our transport protocol survey it is clear that XTP has the widest range of functionality. It offers a rich set of communication syntaxes usable by applications ranging from simple file transfer to distributed computing systems to real-time control systems. XTP also offers a unique multicast facility which can be applied to distributed systems and real-time

applications. In addition, XTP offers the possibility for full hardware implementation which will allow the processing of the full bandwidth of FDDI in real-time [6].

Our performance analysis of UVa XTP 3.5 yielded some interesting results and validated many of the fundamental design concepts of XTP. The throughput and *round-trip* latency performance of XTP was excellent and the XTP memory transfer driver was shown to be extremely efficient in delivering a high percentage of the performance of the XTP engine. XTP's multicast facility also proved to be a high-performance and efficient group communications facility.

Our comparison of XTP to TCP indicated that XTP provided superior throughput performance and greatly superior delay characteristics when connection setup time is included. XTP's *one-way* latency was inferior to that of TCP only because of the communication syntax employed by the memory transfer driver rather than any inherent problem with the XTP protocol.

### 4.4. Future Work

There are several areas of XTP performance which still need to be investigated thoroughly. An investigation of the performance of XTP on a high speed network such as FDDI would yield results of great interest, particularly to the SAFENET community. Extensive performance evaluation of XTP's multicast facility needs to be done, especially for multicast groups of 1-to-10 or larger. Finally, further comparison of XTP to other transport protocols, such as TP4 or other implementations of TCP, would be very valuable in validating the design concepts employed in XTP.

# References


1 U. Black, *OSI — A Model for Computer Communications Standards*, Prentice-Hall, Englewood Cliffs, New Jersey, 1991

2 D. Cheriton and W. Zwaenepoel, "Distributed Process Groups in the V Kernel", *ACM Transactions on Computer Systems*, Vol. 3, No. 2, May 1985.

3 D. Cheriton, "VMTP: A Transport Protocol for the Next Generation of Communications Systems", *IEEE Computer Communications Review*, Vol. 16, No. 3, 1986.

4 D. Cheriton, *VMTP: Versatile Message Transaction Protocol — Protocol Specification*, Stanford University, February 1988. Version 0.7.

5 D. Cheriton and C. Williamson, "VMTP as the Transport Layer for High-Performance Distributed Systems", *IEEE Communications Magazine*, June 1989.

6 G. Chesson, "The Protocol Engine Project", *Unix Review*, September 1987.

7 P. Cocquet, "GAM-T-103 Reference Model for Military Real-Time Local-Area Networks (MRT-LAN)", *Proc. IFIP Workshop on Protocols for High Speed Networks*, Zurich, Switzerland, May 1989.

8 D. Comer, *Internetworking with TCP/IP*, Prentice-Hall, Englewood Cliffs, New Jersey, 1988.

9 *XTP 3.4 User's Manual*, Computer Networks Laboratory, University of Virginia, November 1990.

10 *GAM-T-103 — Military Real Time Local Area Network (Transfer Layer)*, Delegation Generale Pour L'Armement, Ministere De La Defense, Republique Francaise, February 1987.

11 W. A. Doeringer, D. Dykeman, M. Kaiserswerth, B. W. Meister, H. Rudin and R. Williamson, "A Survey of Light-Weight Transport Protocols for High-Speed Networks", *IEEE Transactions on Communications*, Vol. 38, No. 11, November 1990.

12 J. C. Fenton, *User Interface for Xpress Transfer Protocol: Design and Analysis*, University of Virginia, January 1991.

13 "Information Processing Systems — Open Systems Interconnection - Basic Reference Model", *Draft International Standard 7498*, October 1984.

14 "Information Processing Systems — Open Systems Interconnection - Transport Protocol Specification", *Draft International Standard 8073*, June 1984.

15 J. Postel, Ed., "Transmission Control Protocol — DARPA Internet Program - Protocol Specification", *Request for Comments 793*, September 1981.

16 *Xpress Transfer Protocol Definition: Revision 3.4*, Protocol Engines, Inc., Santa Barbara, California, July 1989.

17 *Xpress Transfer Protocol Definition: Revision 3.5*, Protocol Engines, Inc., Santa Barbara, California, August 1990.

18 M. T. Rose, *The Open Book — A Practical Perspective on OSI*, Prentice-Hall, Englewood Cliffs, New Jersey, 1990

19 R. Sanders, *The Xpress Transfer Protocol (XTP): A Tutorial*, University of Virginia, January 1990.

20 R. Simoncic, A. C. Weaver and M. A. Colvin, "Experience with the Xpress Transfer Protocol", *Proc. of the 15th Conference on Local Computer Networks*, Minneapolis, Minnesota, October 1990.

21 W. Stallings, *Handbook of Computer Communications Standards, Volume 1: The Open Systems Interconnection (OSI) Model and OSI-Related Standards*, Macmillan, Inc., New York, New York, 1987.

22 W. Stallings, *Handbook of Computer Communications Standards, Volume 2: Local Network Standards*, Macmillan, Inc., New York, New York, 1987.

23 W. Stallings, *Handbook of Computer Communications Standards, Volume 3: Department of Defense (DOD) Protocol Standards*, Macmillan, Inc., New York, New York, 1988.

24 W. Stallings, *Local Networks*, Macmillan, Inc., New York, New York, 1990.

25 W. Stallings, *Data and Computer Communications*, Third Edition, Macmillan, Inc., New York, New York, 1991.

26 "Survivable Adaptable Fiber Optic Embedded Network I - SAFENET I", *MIL-HDBK-0034 (Draft)*, January 1990.

27 "Survivable Adaptable Fiber Optic Embedded Network II - SAFENET II", *MIL-HDBK-0036 (Draft)*, January 1991.

28 A. S. Tanenbaum, *Computer Networks*, Second Edition, Prentice-Hall, Englewood Cliffs, New Jersey, 1988.

29 *TMS 380 Adapter Chipset User's Guide Supplement*, Texas Instruments, 1986.

30 *WD8003EB High Performance Ethernet PC Adapter with Boot ROM Capability — Engineering Specification - Revision X0*, Western Digital, June 1988.

31 *WIN/TCP for DOS Installation and User Guide — Release 4.0*, The Wollongong Group, Inc., Palo Alto, California, 1989

32 *WIN/API for DOS Programming Guide — Release 4.0*, The Wollongong Group, Inc., Palo Alto, California, 1989