Is XTP Suitable for Distributed Real-Time Systems?

W. Timothy Strayer, Alfred C. Weaver

Computer Science Report No. TR-92-02
January 17, 1992

Is XTP Suitable for Distributed Real-Time Systems?[1]

W. Timothy Strayer
Alfred C. Weaver

University of Virginia
Charlottesville, Virginia
wts4x@virginia.edu
acw@virginia.edu

Abstract

XTP is already recognized as a transport layer protocol for next-generation distributed systems; in this paper we examine the Xpress Transfer Protocol's suitability for distributed *real-time* systems. Distributed real-time systems require a high degree of functionality as well as performance from their communication subsystem. In distributed real-time systems performance gains are typically made at the expense of service functionality. As a consequence many communication subsystems supporting real-time applications are based on MAC layer services. XTP promises high performance at the transport layer through efficient design and an eventual VSLI implementation, as well as a high degree of functionality, much of which is useful for distributed real-time systems. We discuss the requirements on a subsystem in order to support communication in the real-time environment. We examine the features and functionality of XTP. Finally, we conjecture about how XTP meets the requirements of a distributed real-time system, and where it fails to do so.

1. Introduction

The concept of a "computer" is being redefined as processing becomes more distributed. Resources, including the computation server, are not residing in one machine, much less one room, but rather, they are being distributed geographically as dictated by physical and economic reasons. The advent of communication technologies, specifically *networking*, is allowing this to happen. As real-time systems also become less constrained

[1]This work is supported in part by the U.S. Office of Naval Research under contract number CS-DOD/ONR-5030-91. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing official policies, either expressed or implied, of the Office of Naval Research, Naval Ocean Systems Center, Protocol Engines, Inc., or the U.S. Government.

by geography, more emphasis must be placed on the underlying subsystem which provides the interconnection and communication.

Distributed systems rely on the services of the communication subsystem to relieve the application of the concerns of data delivery. The encapsulation of the communication services into a communication subsystem makes transparent to the applications such issues as message length, internetwork topology, and reliable, in-order message delivery. From the ISO OSI Reference Model [1] point of view, such functionally robust communication subsystems include at least the transport and supporting layers.

However, providing communication services to distributed *real-time* systems, where communication is constrained by time, requires a reevaluation of the services currently available, including the algorithms within and interfaces to these services. Since real-time applications are time-constrained, time becomes a resource which requires management. Service guarantees must accompany service requests. Yet, the encapsulation of communication functionality is no less a concern or requirement in real-time systems as it is in general purpose systems.

The Xpress Transfer Protocol (XTP) [2] is a transfer layer[2] protocol designed to meet the communications needs for next generation distributed systems. XTP offers protocol algorithms and procedures which were specifically designed for applications requiring high-speed, low-latency communication services, while not sacrificing the robust functionality characteristics of a transport layer-based communication subsystem. As such, XTP has generated a great deal of interest in such application areas as avionics systems [3], naval systems [4, 5], and space systems [6]. Since a specific design goal of XTP is for its algorithms to be implementable in VLSI, the chip-based XTP is expected to provide nearly

[2]The *transfer* layer refers to the coalescence of the functionality of both the transport and network layers of the ISO OSI Reference Model.

MAC-layer performance at a transport layer interface [7]. XTP is not designed to be a "real-time protocol" *per se*; rather, it is recognized that the functionality and performance inherent within XTP, augmented by other services specifically designed for real-time systems, may be useful in providing support for communications in real-time environments.

This paper examines the issues concerning communications within a distributed real-time system, and how one protocol, namely XTP, addresses some of those issues. We point out the features in XTP which are appealing to real-time applications. We also point out that XTP, in and of itself, will not solve all of the problems concerned with communications within real-time systems, and offer a list of some of the features that a protocol must exhibit if it were truly a protocol for distributed real-time systems.

2. Communications in Distributed Real-Time Systems

Distributed real-time systems most often provide predictability by starting with availability. Dedicated resources, especially the physical interconnection, can provide predictable performance since use of the resources is tightly controlled. Point-to-point wiring provides guaranteed availability of bandwidth and *a priori* knowledge of the communication characteristics, such as latency. Unfortunately, this solution does not scale gracefully since the number of dedicated wires increases combinatorially with the number of interconnected nodes.

Local Area Networks (LANs), such as the IEEE 802 suite [8-10], the ANSI Fiber Distributed Data Interface [11], and the SAE High Speed Ring Bus [12] provide a high-speed shared physical interconnection; the LAN is a shared rather than dedicated resource, so the issue of scalability is traded for the issue of contention. The Media Access Control (MAC) protocols which are part of LANs provide various solutions for resolving

contention, some including prioritized access. Contention resolution characteristics, along with the fact that messages must fit within a fixed-size data frame, allow some of these protocols to be classified as *deterministic*.[3] However, MAC layer services provide only a partial solution to providing communications within real-time systems; the functionality provided at this layer requires applications to provide their own reliability, routing, and message length independence.

The transport layer of the ISO OSI Reference Model classically provides for reliable, end-to-end delivery of arbitrarily long messages over an arbitrary internetworking topology. The issues of data delivery are encapsulated in the functionality of the transport and supporting layers, and the details of how this functionality is provided are transparent to the transport service user. The generality which comes with such robust functionality, however, is problematic for real-time systems; complexity is generally considered the antithesis of predictability.

Consider the two most ubiquitous transport protocols, ISO Transport Protocol class 4 (TP4, [13]) and the Transmission Control Protocol (TCP, [14]). In each case the algorithms were designed to provide a completely reliable connection-oriented service. There are many real-time applications where this reliable data delivery paradigm is inappropriate. There is no concept of time or time-constrained service. In addition, even the message discrimination mechanism is weak: only two levels of priority are offered, with no insurance that this priority will be imposed upon the supporting services. Perhaps most damning is that the transport data delivery is at the mercy of the network layer routing service. Performance guarantees provided at the transport layer can only be as good as those received by the transport layer from its supporting layers.

---

[3]Le Lann [15] claims that this determinism is only applicable under error-free conditions, and thus does not apply when network reconfiguration procedures must take place.

3. Real-Time Communication Requirements

In general, computing systems must maintain two properties: *safeness*, (nothing wrong can happen), and *liveness* (something good will eventually happen). *Real-time* systems add a third property: *timeliness* (things will happen in time for them to be useful) [16]. If we have bounded services and well known process profiles, we can statically examine any system to determine if it will maintain the timeliness property. Unfortunately, systems are usually too complex and the services too difficult to accurately bound to provide a good basis for static analysis. Particularly difficult are communication services.

There is really only one requirement for the communication subsystem—the services provided to the user are appropriate to support the real-time processing being conducted by the system on behalf of the user. It is important to note that the communication subsystem itself does not have to be real-time; rather, it must provide services which support real-time processing. Since the use of the communication services is part of the total execution time for a process, the cost of using the services must be known before a commitment is made to use them. Yet, this requirement states more than that the message delivery time be bounded; rather, it states that the services be appropriate, which enjoins the service provider to offer service paradigms specifically useful for communication in real-time system. Here we offer four points:

1. The subsystem must offer flexible communication paradigms.
2. The subsystem must offer flexible degrees of reliability.
3. The subsystem must offer appropriate message discrimination.
4. The subsystem must offer performance guarantees, specifically with regard to data delivery latency.

A communication paradigm is the expected packet exchanges which implement the information exchange. For example, the *connection-oriented* paradigm of TP4 and TCP implies that data packets are sent in one direction while acknowledgement packets are sent in the other. Distributed real-time systems, by virtue of the prevalence of client/server

relationships and the use of remote procedure calls, in general use a *transaction* paradigm [17]. The transaction paradigm implies that a request message is sent to a server, where it is accepted and a reply generated. The server then sends in return a reply message, often carrying the result of the requested computation. Another important paradigm is a *datagram*, where the information transfer is in only one direction. It is impractical to employ a separate protocol for every communication paradigm required. It is equally undesirable to have one protocol impose a single paradigm on all communication.

Reliability, although often coupled with the communication paradigm, is really an orthogonal issue. Classic connection-oriented paradigms implicitly connote a completely reliable data transfer, while datagrams imply an unreliable service. Real-time systems have use for a wide range of degrees of reliability. In some circumstances, recovering from lost information may actually be harmful to a real-time system, while in other circumstances, reliable delivery of the data is essential. The communication subsystem must not impose a degree of reliability upon the service users; the users must be able to select the degree of reliability appropriate for its application.

A cornerstone of many systems is the notion of *scheduling*, where tasks within the system relay information to a scheduler and are ordered to meet a system-wide criteria. Real-time systems typically include time or some representative of time to help rank the tasks. For instance, the deadline is used in nearest deadline first scheduling, and the inverse of the task's period is used in rate monotonic scheduling. Since tasks are in essence large repositories of state information, maintaining several various task attributes for use in scheduling is easy. Messages, however, are much more constrained about the scheduling information they can carry. It is essential that the task employing the communication subsystem convey enough information to allow the messages to be ordered, yet that information must be concise enough to fit within the message format. Typically the priority

field in a packet is small, perhaps several bits. If the priority field must contain timing information, this width may be too small. Since real-time systems depend on time, it is also important to include some form of timing information in the message discrimination mechanism.

Finally, since real-time systems typically assume worst-case execution times, the communication subsystem must be able to provide the user with worst-case performance guarantees. Ferrari considers the relationship between the service user and the service provider as a legal contract [18], with each party having rights and responsibilities. Among the responsibilities of the user is to provide adequate information about the type of communication required; if the service provider accepts this request, it is responsible for the request being honored. Pivotal to this fourth requirement is the concept of a *guarantee*, that is, the real-time system must be able to use these service guarantees to make more global scheduling decisions.

4. The Xpress Transfer Protocol

In 1987 Greg Chesson[4] undertook to create a transport layer protocol with several properties: that it include the best ideas of existing standard and experimental protocols, that it include network layer routing capabilities, that the algorithms be designed for VLSI implementation, and that it provide clean, regular mechanisms for service without mandating a use or paradigm for that service [19]. The Xpress Transfer Protocol is the result of this effort. XTP provides mechanisms for communication upon which users may implement a wide variety of policies and paradigms. By offering a set of orthogonal mechanisms, the user is provided a functionally rich yet efficient matrix of data transfer services.

---

[4]Chief Scientist at Silicon Graphics, Inc., Mountain View, California, and co-founder of Protocol Engines Inc., Santa Barbara, California.

4.1. XTP Design

XTP provides a powerful mechanism, called an *association*, upon which can be built many communication paradigms. An association is simply the maintenance of state information for a communication between two or more endpoints. When one endpoint decides to begin an association with one or more other endpoints, it initializes state variables, called a *context*, for use in maintaining the state of the association. The initiating endpoint issues a single packet to the other endpoint(s); this single packet exchange is all that is required for each receiving endpoint to set up a corresponding context, and thus establish the association. Furthermore, the association start-up packet is also a data-bearing packet, so a full packet's worth of data may be delivered at the same time the association is being established. No return acknowledgement packet is required for association establishment since reliability is an orthogonal issue.

A fundamental premise of XTP is to separate policy from mechanism, especially with respect to communication paradigms and the error recovery facility. The user of the communication subsystem knows the paradigm most appropriate for its application. While TCP, TP4, and even many experimental transport protocols impose a paradigm upon their users, XTP provides the flexibility necessary to allow the application to choose its paradigm. Furthermore, XTP does not impose a particular error recovery scheme upon its users. Specifically, XTP provides the mechanisms which allow a range of error recovery from none to complete.

Another premise is that the facilities within XTP are orthogonal. The communication paradigm does not impose or assume an error detection policy. XTP derives is flexibility and functionality by allowing the user to choose error, flow, rate, and association control parameters; there are relatively few cases of interaction between these control facilities.

Another design goal of XTP is *flow-through* packet processing. The fields of the packets are placed in the header and trailer of the packet according to how and when the information within these fields is to be processed. Packet parsing information, such as what kind of packet this is, its context identifier, and the various modes, flags, and processing options are placed in the header for immediate access upon packet arrival. The data integrity check field is placed in the trailer since the value for this field depends on the packet's contents. Software implementations of transport layer protocols are able to manipulate the packet in memory segments, and therefore field placement is not as crucial (witness the placement of the checksum field in TCP and TP4 packet headers). Protocols destined for VLSI implementations, where the packet processing may be done as the packet "moves" through the hardware circuits, have the opportunity to place fields so as not to hinder this flow.

XTP is also designed so that packets can be processed "in real-time." This is to say that VLSI implementations of XTP (the so-called *Protocol Engine*, or PE) will be able to parse and process an incoming packet in the time it takes for a packet to arrive. As the PE receives a packet, the addressing information is parsed and the appropriate context is located. Then state information for that association which is maintained in the context is loaded into the XTP logic circuits. As this is occurring, the data within the packet is buffered and the integrity of the data and of the header fields is validated. If the packet passes validation, the protocol commits to accepting this packet. A new packet may arrive immediately following this packet, and, due to concurrency planned within the protocol's VLSI implementation, parsing of this new packet may begin as the processing of the old packet is completing.

4.2. Functionality

XTP provides full transport layer functionality—a reliable, end-to-end delivery of arbitrarily long messages over an arbitrary internetwork topology. Yet, XTP provides functionality in addition to the classic ISO transport layer, much of which is especially useful for communications within real-time systems. In particular, datagrams and transactions are not treated as exceptions but rather natural inclusions within the range of possible transfer syntaxes. Since XTP decouples the notions of paradigm and reliability, the application may choose both the appropriate communication paradigm as well as the appropriate degree of reliability for that communication. XTP's error control facility permits selective retransmission as well as more traditional "go-back-n" retransmission policies. Since XTP is a transfer layer protocol, routing services are included. Multicast, the simultaneous delivery of data to multiple receivers, is a natural extension to the one-to-one unicast association. There is also a facility for the transfer of out-of-band data, that is, data which is tangential to the normal data stream. Finally, XTP provides a 32-bit priority field, called the *sort* field, to convey the message's importance during message processing.

Datagrams and Transactions

A *datagram* is a single message sent from one endpoint to another. Classically, this service is called "connectionless" since the overhead of maintaining a reliable connection is not present; thus a datagram is associated with a "best-effort" delivery. In XTP, an association can be established by the exchange of a single packet. All of the structures necessary to maintain the state of the association are constructed as a result of this single packet, called the FIRST packet. Since this packet may also carry data, this single packet can be treated as a datagram by setting the packet that the End Of Message (`EOM`) and End Of Association (`END`) flags in the packet header. No other packets need be exchanged. Since all of the state structures are built as a consequence of this first packet, however, XTP

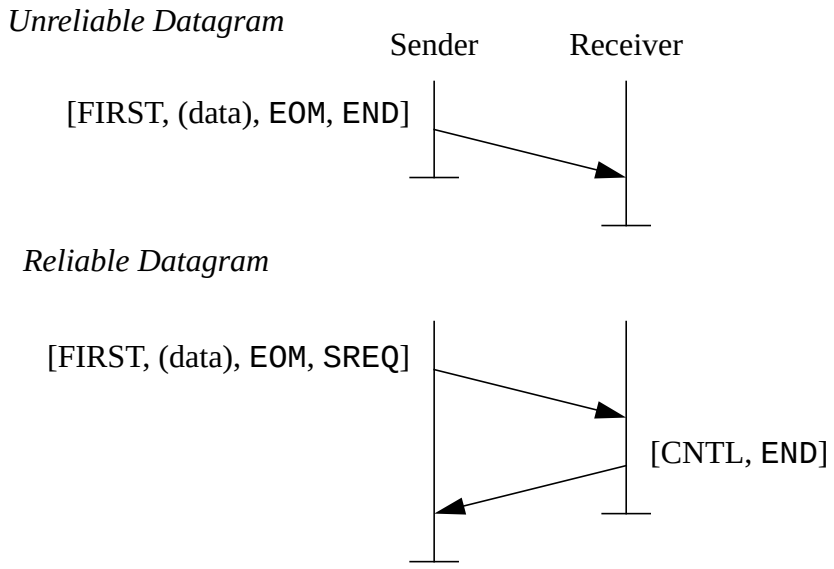

Figure 1 — Datagram Packet Exchange

can also offer a reliable datagram. The datagram becomes reliable when the transmitter sets the Request for Status (SREQ) flag, causing the receiver to reply with a status packet, called the CNTL packet. Figure 1 shows both an unreliable and reliable datagram.

A *transaction* is a two way communication of information in a request/response fashion. One endpoint, often called a client, sends a request message which initiates a transaction. The receiving endpoint, often called the server, processes the request and sends a response. XTP supports transactions as a natural sequence of packet exchanges within an association since associations are by default full duplex. Data sent in the FIRST packet is processed at the server. The server compiles its response into a return packet, called a DATA packet. Note that both the FIRST and DATA packets are data-bearing, and that the receipt of the DATA packet “acknowledges” the receipt of the FIRST packet. If the transaction is unreliable, this return data packet may have the END flag set; if it is reliable, then the SREQ flag is set and the association ends with the status packet. Figure 2 shows both of these exchanges.

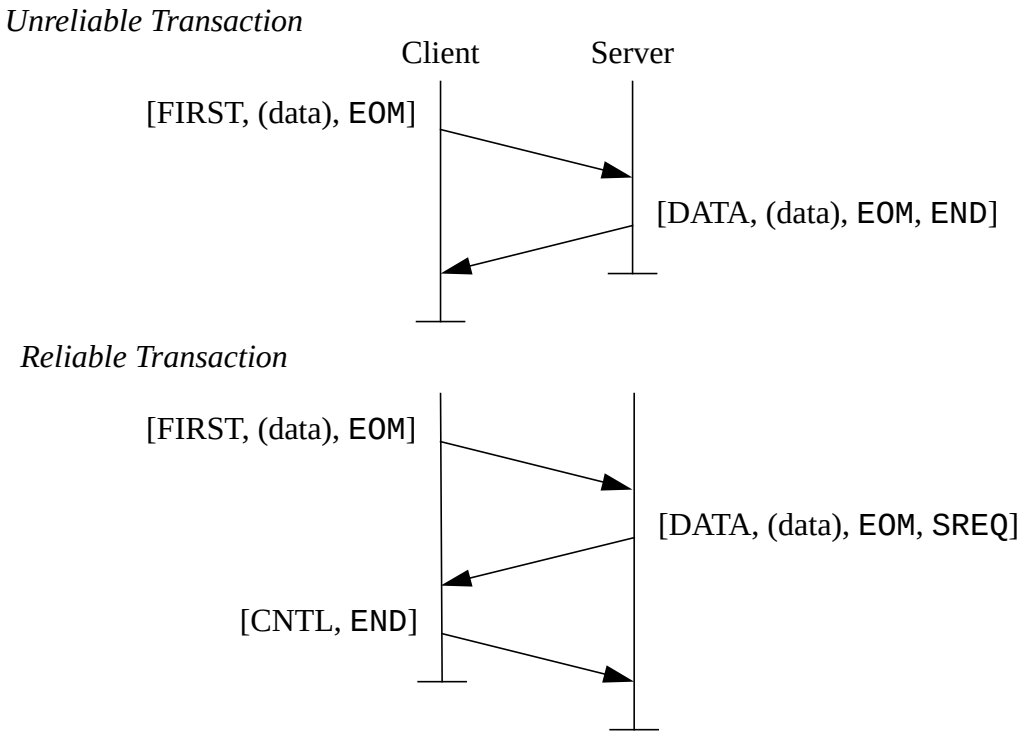

Figure 2 — Transaction Packet Exchange

Routing

XTP is designed to be both a router and an endpoint for an association. Given an arbitrary topology of network segments, a *path* is defined as the series of XTP nodes a packet must pass through from one endpoint to the other. When a FIRST packet is sent, it is given a unique path identifier. The FIRST packet cuts a path through the intermediate nodes, leaving a trail of such path identifiers, so that any subsequent packet in either direction may trace the path between the two endpoints. Note that this eliminates the need for a full address in each packet; after the FIRST packet cuts the path, the subsequent packets need only know the path identifier to use that path.

Reliability

Reliable data delivery is, of course, provided by XTP. However, XTP recognizes that reliability is an orthogonal issue to the paradigm of communication employed, and that applications require varying degrees of reliability. Traditional connection-oriented protocols like TCP and TP4 impose complete reliability along with the connection paradigm. In XTP, the user may specify that the communication be conducted in “no error mode,” where lost data are not retransmitted. To effect this, the transmitter sets the “no error” bit flag in the FIRST and subsequent packets, instructing the receiver to always report that no data has been lost. This does not imply that CNTL packets are never requested or sent; on the contrary, CNTL packets serve purposes other than acknowledgement. What “no error mode” allows is the continued progress of data transfer without halting to recover from lost data, as may be appropriate for data streams consisting of periodically sampled data (e.g., sensor values).

Selective Retransmission

Since the transmitter is responsible for supplying the receiver with any data that is known or suspected to be lost, the XTP receiver can provide the XTP transmitter with very specific data delivery information. The receiver keeps track of data contiguity, and as gaps in the data arise, the receiver builds a list of data *spans* (correctly received data). When asked for delivery status by the transmitter, the receiver places this span information into the CNTL packet. Thus, the information is available to the transmitter that will allow it to selectively retransmit only that data which is missing. Since selective retransmission is not always a benefit, the transmitter may ignore this span information and simply “go-back-n”, retransmitting data from the last contiguously received byte of data.

Multicast

Multicast is the simultaneous delivery of messages from a transmitter to more than a single receiver. The transmitter along with the receiver set is called the *multicast group*. Multicast eliminates the need to set up a separate unicast communication with each receiver, and allows the transmitter to communicate with multiple peers without an extension to the communication paradigm. Several issues pervade multicast, especially a transport layer multicast, including group membership, reliable delivery, and the maintenance of a full duplex channel. If reliable delivery of the data were not essential, the membership of the multicast group could be dynamic without affecting the transmission. However, if the delivery must be reliable, the transmitter must be aware of the states of all of the members of the multicast group, and dynamic group membership impedes this. The problem of full duplex communication, or *concentration*, includes the issue of whether a return data stream actually makes sense (consider, for example, a file being concentrated to one endpoint).

XTP provides a multicast service. In fact, it is a *semi-reliable* multicast service in the sense that, as long as receivers are active members of the multicast group, the receivers may request retransmission of lost data. The service is semi-reliable since the transmitter can not monitor whether the full receiver set is active. Since the transmitter is relieved of this responsibility, it makes sense to allow receivers to dynamically join and leave the multicast group as long as, upon joining the group, the receiver does not ask for a retransmission of data prior to the first data it receives. The entire multicast transmission is not contingent upon the health of any one member of the group, nor does this multicast need to be “brought down” in order for a failed node to rejoin the group. This allows a redundancy which is particularly useful for fault tolerance.

XTP treats multicast communication as a natural extension to the association. There is a flag bit, however, which indicates to the receivers that they are part of a multicast communication and therefore should employ acknowledgement schemes which are appropriate for this situation.

Out-Of-Band Data Delivery

XTP provides a facility for out-of-band data delivery. When an appropriate flag indicates thus, a field of 8 bytes is inserted prior to the normal *data* field. This field is called the *btag* field, for beginning tag, since the field precedes other data and the contents are "tagged" with special meaning. XTP does not examine or use the contents; this field is for end-to-end delivery of data other than the normal data stream. Such out-of-band data is useful for circumstances where the normal data stream has attributes or control information associated with segments of it, e.g., a file name associated with the file data, or a timestamp for sampled data.

Message Priority

The XTP protocol specifies that at any decision point, processing will be applied to the highest priority packet (or context with the highest priority packet) ready for processing. The priority is conveyed by a 32-bit field called *sort*. The user specifies the *sort* value of a message when the service call is made to the XTP implementation. The context handling this message segments the message into packets and inserts the *sort* value into the sort field of each packet. The active contexts are serviced in priority order, which is from low *sort* value to high. At each processing point along the path (that is, at any routers and at the destination) incoming packets are ordered for attention by using this *sort* value. Finally, the assembled message, along with its *sort* value, is delivered to the destination user (of course,

messages are delivered in *sort* order as well). Messages with no *sort* value are processed only after all other messages are processed.

The *sort* value is 32-bits wide to allow a variety of priority schemes to be used. XTP does not impose a scheme; rather, XTP processes in ascending *sort* value order. The user, on the other hand, may assign meanings to the 2^{32} possible *sort* values. Perhaps more useful is the interpretation of the *sort* value as a timestamp representing a deadline. Since processing is done in ascending *sort* order, the nearest deadlines are processed first.

5. XTP and Communications within Real-Time Systems

In sections 2 and 3 we discussed the communication subsystem and what is required of it for use within a distributed real-time system. In section 4 we discussed the Xpress Transfer Protocol, with emphasis on features and functionality that make it useful as a component of a communication subsystem that supports real-time applications. This section will conjecture about how XTP may be used within a communication subsystem which supports real-time communication. We examine how XTP meets the requirements set forth in section 3. Where XTP or an XTP-based communication subsystem fails to meet the requirements we discuss what might be needed within a protocol or the subsystem to meet the need.

Flexibility with regard to communication paradigms

XTP's separation of mechanism from policy, especially with respect to the transfer syntax allowed by the association, and XTP's decoupling of the various control facilities, especially reliability from communication paradigm, indicate that XTP can satisfy the first two requirements given in Section 3. Yet XTP offers in essence only two degrees of reliability, completely reliable or "no error mode." It would be useful to have a service where a parameterized amount of error can be tolerated over periods of time [20]. For

example, packetized voice data can suffer intermittent loss, yet a burst loss would degrade the service sufficiently to require some form of recovery. Furthermore, the delivery of a message should not be contingent upon whether or not an attempt was made to recover from lost data. Even if acceptable data loss occurs, the receiver should be able to attempt recovery without the message's delivery being blocked while awaiting the arrival of the retransmitted data.

Flexible degrees of reliability

Ideally, the task employing the communication subsystem should pass all of the information necessary to impress upon the communication subsystem exactly how important this particular communication is. If this impression could be made, the communication subsystem could schedule the messages according to their importance to the system. Yet, it is not clear that all of the task attributes used to schedule tasks are useful to the communication subsystem, nor is it clear which of the task's attributes should be included in the request to the communication subsystem. Instead of guessing, XTP offers simple, straight-forward priority queued processing but does not impose an interpretation of this priority. This priority, or *sort* value, is used to order all message processing from smallest *sort* value to largest. The *sort* field is wide enough to support a large number of interpretations. For example, if the *sort* value is interpreted as a deadline, then messages are handled in nearest deadline order. Also, the *sort* value is used at every processing point along the path from the sending user to the receiving user.

Appropriate message discrimination mechanisms

As wide as the *sort* field is, however, it is still not clear that a single field imposing a monotonically increasing order on processing adequately encapsulates all of the timing and other information which indicate how important the message is [21]. In particular, there

is no way to include both a deadline and a criticalness measure within the *sort* field. The ability to *schedule* messages is traded for a simple, straight-forward ordering mechanism. As real-time systems expand scheduling policies to include more aspects of the tasks and the state of the system, this simple mechanism may prove inadequate.

Performance guarantees

XTP does not, however, offer any performance guarantees such as latency control. XTP does not interpret the *sort* value, but rather uses the value for ordering message processing in a priority queueing discipline. It is well known that priority queues may cause unbounded delays for all but the highest priority customers; XTP does not attempt to bound message latency. Consequently, an XTP-based communication subsystem can not make definite guarantees about service delays.

6. Conclusions

XTP represents a major advance in providing a transport protocol which is of utility to a wide range of real-time systems; its multicast capability and its internal priority operation are particularly valuable. Multicast allows any number of receivers to participate in a reliable one-to-many transmission, thus conserving bandwidth; the operation of the priority mechanism (*sort* field) guarantees that, with a granularity of one packet's transmission time, every element of the end-to-end delivery subsystem (including interior routers) is working on its most important packet. We believe that these two features will prove to be of critical importance to modern real-time distributed systems.

Even so, the definition of XTP is not a perfect match to the needs of real-time systems. Two areas could still be improved:

1. The priority system is static, rather than dynamic, so if a packet is sufficiently long-lived that its importance may change over time, that change can not be represented.
2. XTP does not deal directly with the issue of network global time. XTP can be used by a network time protocol, and messages can be time-stamped by the application process (using the “tagged data” field), but XTP itself does not support the concept of global time or base any decisions on the global time

The advantages and disadvantages of XTP will only emerge as XTP begins to be used in the design of actual real-time systems.

7. References

[1] International Standards Organization, “Information Processing Systems - Open Systems Interconnection - Basic Reference Model,” *Draft International Standard 7498*, October 1984.

[2] Protocol Engines, Inc., “XTP Protocol Definition, Rev 3.6,” Protocol Engines Inc., December 1991.

[3] Cohn, M. D., “A Proposed Local Area Network for Next-Generation Avionic Systems,” *Proceedings of NAECON*, Dayton, Ohio, (May 23, 1988).

[4] Cohn, M. D., “A Lightweight Transfer Protocol for the U.S. Navy Safenet Local Area Network Standard,” *Proceedings of the 13th Conference on Local Computer Networks*, Minneapolis, Minnesota, pp. 151-156, (October 10-12, 1988).

[5] Marlow, D. T., “Requirements for a High Performance Transport Protocol for Use on Naval Platforms - Revision 1,” Working Papers, NSWC, (7/23/89).

[6] Weaver, A. C. and Simoncic, R., “Communication for the NASA Space Station,” *Proceedings of the 14th Conference on Local Computer Networks*, Minneapolis, Minnesota, pp. 333-339, (October 10-12, 1989).

[7] Chesson, G. and Green, L., “XTP-Protocol Engine VLSI for Real-Time LANs,” *Proceedings of the Sixth European Fibre Optic Communications and Local Area Networks Exposition*, Amsterdam, Netherlands, (June 29-July 1, 1988).

[8] Institute of Electrical and Electronics Engineers, “IEEE Standard 802.3 Carrier Sense Multiple Access with Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications,” 1985.

[9] Institute of Electrical and Electronics Engineers, “IEEE Standard 802.4 Token-Passing Bus Access Method and Physical Layer Specifications,” 1985.

[10] Institute of Electrical and Electronics Engineers, “IEEE Standard 802.5 Token Ring Access Method and Physical Layer Specifications,” 1985.

[11] American National Standards Institute, “FDDI Token Ring Media Access Control Standard,” *Draft proposed Standard X3T9.5/83-16, Rev. 10*, February 1986.

[12] Society of Automotive Engineers, “SAE AS4074.2 High Speed Ring Bus, Final draft Standard,” June 1987.

[13] International Standards Organization, “Information Processing Systems - Open Systems Interconnection - Transport Protocol Specification,” *Draft International Standard 8073*, July 1986.

[14] Postel, J. ed., “Transmission Control Protocol — DARPA Internet Program Protocol Specificating,” *RFC 793, USC/Information Sciences Institute*, (September 1981).

[15] Le Lann, G., Meyer, J. F., Movaghar, A. and Sedillot, S., “Real-Time Local Area Networks: Some Design and Modeling Issues,” Institut National de Recherche en Informatique et en Automatique, No. 448, Le Chesnay, France, October 1985.

[16] Le Lann, G., Guth, R., ed., “Distributed Real-Time Processing,” *Computer Systems for Process Control, Proceedings of a Brown Boveri Symposium*, Baden, Switzerland, pp. 69-90, (1985).

[17] Tokuda, H., Mercer, C. W. and Ishikawa, Y., “The ARTS Distributed Real-Time Kernel and its Toolset,” Report, 1989.

[18] Ferrari, D., “Client Requirements for Real-Time Communication Services,” Technical Report Technical Report-90-007, International Computer Science Institute, March 6, 1990.

[19] Chesson, G., “The Protocol Engine Project,” *UNIX Review*, Vol. 5, No. 9, (September 1987).

[20] Dempsey, B. J., W. T. Strayer, A. C. Weaver, “Adaptive Error Control for Multimedia Data Transfers,” *Proceedings of the International Workshop on Advanced Communications and Applications for High Speed Networks*, Munich, Germany, March 16-19, 1992.

[21] Strayer, W. T., Dempsey, B. J. and Weaver, A. C., “Making XTP Responsive to Real-Time Needs,” The University of Virginia, Department of Computer Science Technical Report TR-89-18, November 1989.