Communication Services for Real-Time Systems: An Examination of ARTS over XTP

W. Timothy Strayer

Computer Science Report No. TR-91-34
December 6, 1991

Communication Services for Real-Time Systems: An Examination of ARTS over XTP[*]

W. Timothy Strayer
Department of Computer Science
University of Virginia
Charlottesville, Virginia
strayer@virginia.edu

November 11, 1991

Abstract

ARTS is a distributed real-time operating system designed to provide the user with a predictable, analyzable, and reliable service. The Xpress Transfer Protocol is a next-generation, high performance, high functionality transport layer protocol. We examine the role of the communication subsystem in providing a real-time system communication support. We use the ARTS system and XTP as a case study for understanding the needs of a distributed real-time system and how a transport layer protocol may be able to help meet those needs. Finally, we offer suggestions about how ARTS may be able to take advantage of features in XTP, and how XTP may be augmented to better serve a distributed real-time system such as ARTS.

1. Introduction

The ARTS operating system ([TOKU89], [TOKU90]) is a distributed real-time operating system designed for the Advanced Real-Time Technology (ART) testbed at Carnegie-Mellon University. The goal of the ARTS system is to provide users with a predictable, analyzable, and reliable distributed real-time computing environment. Since the system is distributed, particular emphasis is placed on the communication subsystem,

[*]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 author 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., Carnegie-Mellon University, or the U.S. Government.

especially the needs of the real-time systems and what is required of the communication subsystem to meet these needs.

The Xpress Transfer Protocol (XTP) ([PEI89]) is a transfer† layer protocol designed to meet the communications needs for next generation distributed systems. XTP offers protocol algorithms and procedures which were specifically designed 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 ([COHN88a]), naval systems ([COHN88b], [MARL89]), and space systems ([WEAV89]). 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 MAC-layer performance at a transport layer interface ([CHES88]). XTP is not designed as 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, are useful in providing support for communications in real-time environments.

This paper addresses the role of a communication subsystem from two perspectives: what a distributed real-time system requires of a communication subsystem, and what services a communication subsystem can and should provide in order to support a real-time application. The ARTS real-time operating system and XTP are chosen as case studies. Since ARTS is currently implemented over IEEE 802.5 Token Ring using a communication manager to implement some transport layer functionality, part of this study will be to conjecture about how XTP can be used in place of the current communication subsystem. Finally, several observations will be drawn as to how the ARTS communications primitives

†*Transfer* refers to the coalescence of the functionality of both the transport and network layers of the ISO OSI Reference Model ([ISO7498]).

may be modified to take advantage of the functionality of XTP, and how XTP would have to be modified to more fully provide support to a real-time operating system.

2. Distributed Real-Time Systems

The concept of a “computer” is being redefined as processing becomes more distributed. Resources, including the computation server, no longer reside in a single machine or even a single 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 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 application such issues as message length, internetwork topology, and reliable, in-order message delivery. From the ISO OSI Reference Model ([ISO7498]) 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 requires a reevaluation of the services currently available, including the algorithms within and interfaces to these services. Real-time applications are necessarily constrained by time, and thus predictability is a requirement. 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.

2.1. 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 wire plant increases combinatorially with the number of interconnected nodes.

Local Area Networks (LANs), such as the IEEE 802 suite ([IEEE85a-c],), the ANSI Fiber Distributed Data Interface ([ANSI86]), and the SAE High Speed Ring Bus ([SAE87]) provide a high-speed shared physical interconnection. The Media Access Control (MAC) protocols which are part of the 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*. Le Lann ([LELA85a]) claims that this determinism is only applicable under error-free conditions, and thus does not apply when network reconfiguration procedures must take place. Furthermore, MAC layer services fail to provide adequate solutions to communications within real-time systems since the functionality provided at this layer requires applications to emulate reliability, routing, reliability, and message length independence.

The Transport Layer of the ISO OSI Reference Model does provide for reliable, end-to-end delivery of arbitrarily long messages over an arbitrary internetworking topology, but no extant standard transport protocol also provides a means of including timing information. The ISO Transport Protocol class 4 (TP4) ([ISO8073]) and the DARPA Transmission Control Protocol (TCP) ([DARPA81]) both have only two levels of priority, and neither have a notion of time. Furthermore, both TP4 and TCP are based on a connection-oriented paradigm of information transfer; real-time systems generally require a flexible array of paradigms offered through the services of the communication subsystem, including datagrams and transactions. Of the experimental transport protocols, the Versatile Message Transaction Protocol (VMTP, [CHER88]) and the Xpress Transfer Protocol (XTP) both are designed for support of real-time systems, although neither explicitly provides performance guarantees.

2.2. Real-Time Communication Requirements

In general, distributed systems must maintain two properties: *safeness* (nothing wrong can happen) and *liveness* (something good will eventually happen). Distributed real-time systems add a third property: *timeliness* (things will happen in time) ([LELA85b]). 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. Communication services providing support to real-time applications, therefore, must provide these applications with the ability to specify their performance requirements and to obtain guarantees about the satisfaction of these requirements ([FERR90]).

The communication subsystem must be able to provide a degree of performance appropriate for the application. Contrary to common interpretation, “real-time” does not necessarily imply *fast*; rather, predictable levels of performance, and their availability, are more pressing issues. Certainly high speed, high throughput, and low latency are important metrics to any application. In real-time systems, however, it is more important to guarantee a level of service.

Traditional interfaces to communication services do not allow performance requirements to be conveyed, nor the guarantees returned. TP4 allows quality of service parameters to be negotiated during connection setup, although there are no mechanisms within TP4 itself to guarantee that these parameters will always be met. Indeed, if conditions arise such that the quality of service offered falls below that requested, the action required of the protocol is to notify the protocol user and abort the connection.

3. ARTS

The ARTS distributed real-time operating system is designed to provide users with a predictable, analyzable, and reliable distributed real-time computing environment. This environment is based on an object-oriented paradigm — every computational entity within ARTS is represented as an object. ARTS employs an Integrated Time-Driven Scheduling model to allow the system designer to analyze given task sets for schedulability. Since ARTS is a distributed operating system, a communication subsystem provides information exchange services. This section briefly describes the computational and scheduling models within ARTS, and, in more detail, examines the communication subsystem.

3.1. Computational Model

Each computational entity within ARTS is represented by an *artobject*. One or more *threads*, or lightweight processes, may be declared within the artobject. Also, there are *operations* which may also be declared within the artobject; *public* operations may be invoked by other threads wishing to use them, whereas *private* operations are only for the use of the threads within that artobject.

Artobjects differ from common objects in that the artobject provides a *time encapsulation*. Worst case computation times are associated with each *operation* within an object, providing a “time-fence” for isolating and handling timing exceptions. Threads may have timing attributes associated with them as well. Threads with timing attributes are termed *real-time*; those without are *nonreal-time*. These timing attributes include worst case execution time, period, phase, and delay parameters. In practice this requires the specification of deadlines for periodic and aperiodic hard real-time threads.

3.2. Scheduling Model

The scheduling model within ARTS is the Integrated Time-Driven Scheduling (ITDS) model which can schedule both periodic and aperiodic tasks in an integrated fashion. For hard real-time tasks (periodic tasks and aperiodic tasks with hard deadlines) the ITDS model allows the system designer to predict the schedulability of the given tasks. For soft real-time tasks, the system designer can predict if the worst case response time will satisfy the task's response time requirements. During overload situations, the ITDS model provides control over which tasks should complete their computation.

The processor utilization is first determined for the hard real-time tasks, and the rate monotonic scheduling theory results are applied to determine schedulability to this task set. Since tasks may share resources, and thus be subjected to indefinite priority inversion, the ITDS model includes a priority ceiling protocol for bounding the blocking time; this extension to the rate monotonic theory allows utilization to be computed for more general task sets. Once the hard real-time tasks are guaranteed, the processor capacity remaining is applied to the soft real-time tasks. For example, the Deferrable Server ([LEHO87]), which is a special periodic task, may be used to provide the aperiodic tasks with low response time. Rather than waiting for "left-over" cycles, the Deferrable Server "gives" its processor time to the highest priority task.

The integrated time-driven scheduler is constructed such that the scheduling policies and the mechanisms for implementing them are separate; thus scheduling policies other than the rate monotonic algorithm can be used. Among the policy objects are rate monotonic, earliest deadline first, least slack time, as well as FIFO, round robin, and fixed priority. Any scheduling algorithm can be implemented in a policy object and tested within ARTS; however, for predictability purposes it is important that the schedulable bound be determined.

3.3. Communications in ARTS

A thread may wish to invoke an operation that is within a remote object. The communication necessary for this remote operation invocation is provided by primitives within ARTS (these primitives are themselves operations) which induce a Request-Accept-Reply communication paradigm. The local thread requests that an operation be invoked at a remote object. A thread at the remote object, having previously executed an accept operation, receives this request and invokes the operation. When the operation completes, the remote thread replies with the result.

Message traffic is analyzed for schedulability in a similar manner as are the tasks themselves. Information about the communication patterns among the objects is available *a priori*, including the specification of periodic traffic and the rates for aperiodic traffic. First there is an attempt to guarantee all hard real-time periodic message traffic at the system design stage. Then the aperiodic messages are included such that their response times are minimized. Since messages carry a time fence value, timing faults can be isolated in the same way as with local operation invocation.

Currently, the communication subsystem within ARTS supports several communication primitives by employing a Communications Manager to schedule and dispatch messages. The testbed over which ARTS is implemented uses both an Ethernet and IEEE 802.5 Token Ring for the physical and media access control layers of the network.

3.3.1. Requirements

The requirements for real-time communication subsystem, as derived from [TOKU90], are the following:

- The subsystem should provide a mechanism for handling the timing requirement of the message traffic.
- Since real-time systems rely on fine priority granularity to make guarantees about system performance, this priority granularity should be extended to the communication subsystem.
- The message format must include a priority field.
- The communication subsystem should deliver the priority and timing information along with the message to the remote thread to help in scheduling.

The ARTS communication subsystem attempts to meet these requirements in the following ways. Timing information in the form of the “time fence” is passed to the communication subsystem as a parameter to the communication primitives. This informs the communication subsystem of the timing requirements of the requesting thread. The communication subsystem must make the mapping from task priority to message priority. If, as is often the case, the message priority is not as fine as the task priority, the communication manager must be able to make an appropriate mapping. As for a message priority field, most media access control protocols (except Ethernet) and most transport layer protocols have some form of prioritization carried as part of the message format. It is up to the implementation of the protocols and the communication subsystem as to whether this priority is available to the receiving thread.

3.3.2. Communication Primitives

Since every communication in ARTS is caused by an operation invocation to a target object’s operation, the communication primitives reflect the Request-Accept-Reply paradigm. ARTS currently supports four primitives: **Request**, **Accept**, **Reply**, and **CheckRequest**. These primitives provide a synchronous service, where the caller must block until the reply is received. Asynchronous extension to these primitives include **AsyncRequest**, **AsyncRequestAll**, **GetReply**, and **CheckReply**. For purposes of this discussion both the synchronous and asynchronous primitives will be included.

The primitives and their arguments are given below.

val = **Request**(invocation_dsc, msg_dsc, reply_msg_dsc)
val = **AsyncRequest**(invocation_dsc, msg_dsc, reply_msg_dsc)
val = **AsyncRequestAll**(invocation_dsc, msg_dsc, reply_msg_dsc)
val = **GetReply**(invocation_dsc)
val = **Accept**(invocation_dsc, msg_dsc)
val = **Reply**(invocation_dsc, reply_msg_dsc)
val = **CheckRequest**(operation_result, operation_selector, any, timeout)
val = **CheckReply**(reply_result, reply_selector, any, timeout)

The argument *invocation_dsc* is the argument which holds the invocation descriptor. This descriptor is a structure which contains a transaction identifier, object identifier, thread identifier, operation identifier, the current time fence value, and fields for the network protocol information. The transaction identifier is a unique value set by the communication subsystem upon a request or accept primitive, and used to identify that particular transaction on any subsequent calls to the communication subsystem. The *msg_dsc* and *reply_msg_dsc* are arguments which hold a message descriptor. A message descriptor contains the address of the message, the message size, and the message priority information. The argument *operation_result* indicates the list of requested operations, and the argument *operation_selector* represents a bit map which indicates a list of corresponding operations. Similarly, the argument *reply_result* is a returned bit map of available transaction identifiers, and *reply_selector* is a bit map used to specify a list of transaction identifiers. The argument *any* indicates that the condition depends on *any* or *all* of the specified selector bits. The *timeout* argument specifies how long to wait until the condition is true.

3.3.3. Time Fence

A "time fence" is a value associated with each real-time operation reflecting the worst case execution time allowable for that operation. The time fence provides a run-time check on the timing requirements of an object. The time fence uses information about the worst case time for a computation to set a timer on the actual computation. This value is checked before each object invocation to verify that the slack time exceeds the worst case execution time of the invoked operation. Thus timing faults are quickly discovered and isolated.

For communications, the request message carries the requesting thread's current time and worst case remaining time values. After the request message is received at the remote site, a check is made to determine if the operation can be invoked within the worst-case remaining time of the requesting thread. If this check fails, a Fence Error occurs. If the reply message fails to arrive in time at the requesting thread, the system will timeout and transfer control to an exception handler.

3.3.4. Communication Manager

The Communication Manager (CM) is an object which handles all of the network messages for the local ARTS kernel. The CM is a single point through which all incoming and outgoing communication for the local site must pass, and is therefore both a point of contention and a possible point of priority inversion. The CM encapsulates the underlying implementations and other details of the network service, including which protocols are used and how the task priority and timing information are mapped to the network or message priority.

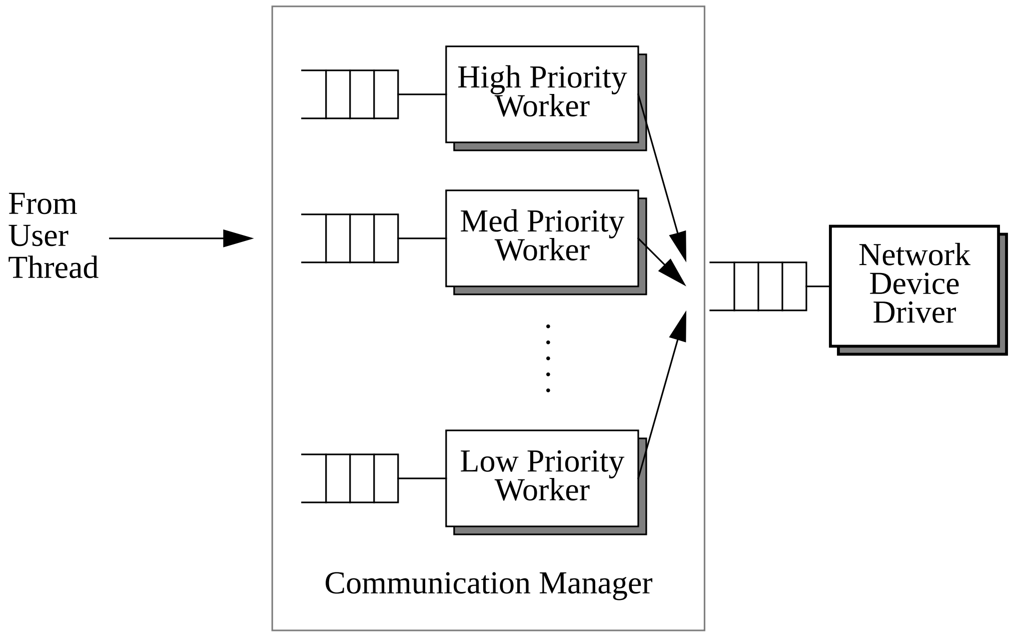

Figure 1 — Communication Manager Organization

The CM is implemented as a single thread which accepts the messages and places them in a prioritized queue. A prioritized queue is used instead of a FIFO queue to avoid priority inversion at this single access point. The CM also has several prioritized worker threads, each with a FIFO queue, as shown in Figure 1. There are as many worker threads as there are network priorities, and these threads are prioritized in a fashion which corresponds to the message priority. Since these are threads, they may be preempted at any time during the processing of a message, which further reduces the priority inversion. From these processing queues the messages are given to the network device driver (the interface to the MAC layer) for delivery.

The processing which occurs within the prioritized workers is roughly equivalent to that which is done within a transport protocol. Therefore, the CM is sometimes referred to as the Real-Time Transport Protocol, or RTP. When a Transport Layer protocol is included within the communication subsystem, the CM looses much of its responsibility.

4. The Xpress Transfer Protocol

In 1987 Greg Chesson undertook the creation of 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 were designed for VLSI implementation, and that it provide clean, regular mechanisms for service without mandating a use or paradigm for that service ([CHES87]). The Xpress Transfer Protocol is the result of this effort. This project has garnered the interest and support of many industry and research institutions, as well as becoming the protocol of choice for the lightweight protocol stack in the Navy SAFENET project ([NOSC91]).

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 some state variables, called a *context*, for use in maintaining the state of the association. The initiating endpoint issues a single packet to the other endpoints; this single packet exchange is all that is required for each receiving endpoint to set up a corresponding context, and the association is established. 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 as the association is being established. Note that acknowledging the association establishment is an orthogonal issue.

A fundamental premise of XTP is to separate policy from mechanism, especially with respect to communication paradigms. The user of the communication subsystem knows the paradigm most appropriate for its application. While TCP, TP4, and even VMTP impose a paradigm upon their users, XTP provides the flexibility necessary to allow the application to choose its paradigm. A datagram may be sent by filling the first packet with data and setting the End Of Message and End Of Association bit flags. A reliable datagram is achieved by setting the End Of Message and Status Request flags; this instructs the receiver to issue a packet containing data delivery status information. Since an association is full duplex, data may be sent in both directions without additional setup. Therefore, transactions can be handled in an association which uses as few as two packets.

Another premise of XTP is that the transmitter drives the receiver insofar as controlling the association. The transmitter requests data delivery status information from the receiver since the transmitter is the entity with the knowledge of what degree of reliability and acknowledgement scheme is appropriate. At association setup, the transmitter sets certain “mode” flags to inform the receiver of some basic properties of the association, including “no error mode”, where delivery errors are ignored, and “reservation mode”, where flow control is based on conservative buffer availability.

A fundamental design goal of XTP was *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 ISO Transport 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 will be able to parse and process an incoming packet in the time it takes for a packet to arrive. As XTP receives a packet, the addressing information is parsed and the appropriate data structure, called a *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 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.

Certainly if an XTP chipset is placed in conjuction with, say, an FDDI chipset, the performance possibilities at the XTP interface could approach those currently attainable only at the FDDI MAC interface. This promise of performance is a major selling point of XTP, and one that makes it appealing to performance-constrained and time-constrained applications. It is certainly advantageous to use high-performance solutions for real-time applications, but, as has been cautioned, performance alone can not guarantee that the requirements of real-time systems are met.

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. In particular, the ability to separate paradigm from mechanism allows XTP to offer datagrams and transactions without requiring either a separate protocol (as is the case with TP4 and TCP), or unnecessary packet exchanges. 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. Finally, XTP provides an extraordinarily wide (32 bits) 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 End Of Message (EOM) and End Of Association (END) flags. No other packets need be exchanged. Since all of the state structures are built as a consequence of this first packet, however, XTP 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 2 shows both an unreliable and a 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

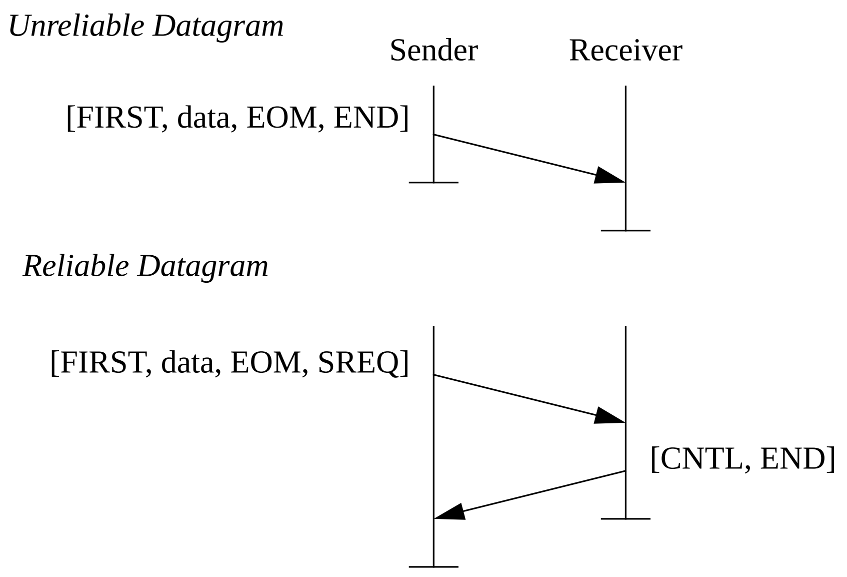

Figure 2 — Datagram Packet Exchange

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 a SREQ flag is set and the association ends with the status packet. Figure 3 shows both of these exchanges.

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, leave 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 use the path identifier to use the path.

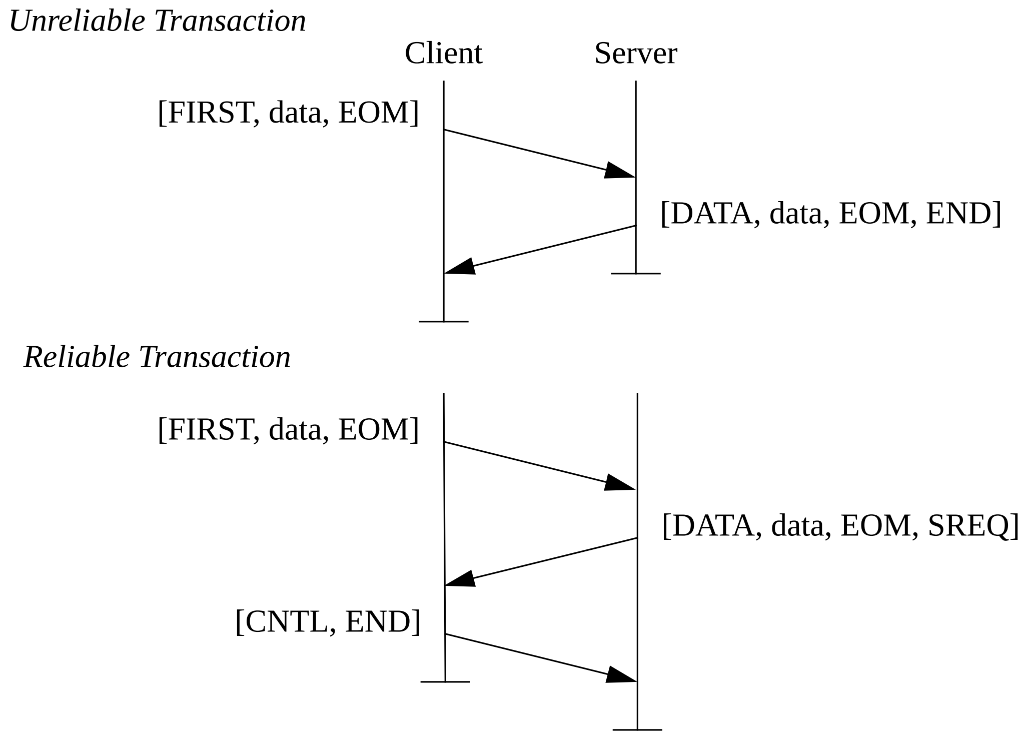

Figure 3 — Transaction Packet Exchange

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 issue 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 if the full receiver set is active. Since the transmitter is relieved of this responsibility, it makes sense to allow receivers to dynamically join and depart 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.

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.

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. When asked for delivery status by the transmitter, the receiver places this gap 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 gap information and simply “go-back-n”, retransmitting data from the last contiguously received byte of data.

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.

4.2.1. Message Priority

The XTP protocol specifies that at any decision point, processing will be done to the highest priority packet (or context with the highest priority packet) ready to receive processing. The priority is conveyed by a 32-bit field called the *sort* field. 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 *sort* values. For example, there are 4 billion separate priority levels. Perhaps more useful is the interpretation of the *sort* value as a timestamp representing the deadline. Since processing is done in ascending *sort* order, the nearest deadlines are processed first.

5. ARTS on XTP

Section 3. discussed the ARTS distributed operating system, with emphasis on the scheduling model and the communication subsystem. Section 4. 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 ARTS. First, we examine how XTP meets the requirements set forth in Section 3.3.1. We look at the primitives, especially the message descriptor argument, for how ARTS can interface with XTP. Next, since ARTS uses the time fence mechanism extensively, we speculate on whether XTP will meet the needs of this mechanism. Finally, XTP offers transport layer functionality while this functionality is implemented as the Communication Manager in ARTS; we examine the issues involved with replacing the Communication Manager with XTP.

Requirements

Recall the requirements of Section 3.3.1. In brief, a real-time communication subsystem must (1) provide a mechanism for handling message timing requirements, (2) provide a prioritization mechanism that is an extension of the system-wide priority scheme, (3) include a priority field in the message format, and (4) deliver the priority and timing information user along with the message to the destination. Below we discuss how XTP meets these requirements.

XTP does not explicitly use the timing requirements of a message for scheduling the processing of that message, as (1) requires. Rather, XTP offers simple, straight-forward priority queued processing. XTP does not impose an interpretation of this priority; if the priority (as embodied in the *sort* field) is a deadline, then messages are handled in nearest deadline order. It is recognized in [STRA89] that a single priority field does not adequately convey all of the timing or other information about how important this message is. Of course, the *sort* field can be divided into two parts, one for the deadline and one for the criticalness, but this approach is user-defined.

Requirement (2) suggests that the message priority be an extension of the task priority. Task declarations are augmented by priority, worst-case execution time, period, phase, and delay parameters. It is not clear that all of these parameters are useful to the communication subsystem; however, it is important that the task convey enough information about the importance of the communication to the communication subsystem. XTP uses the *sort* value to order all of the processing on messages at any processing point during the transfer of the message; XTP does not consider worst-case execution time, period, phase, or delay.

Since the *sort* value is also included in the format of every packet format, the third requirement is satisfied. XTP also makes available to the remote user this *sort* value used for this message. This satisfies requirement (4).

Primitives

Of the arguments to the communications primitives, the invocation descriptor and the message descriptor provide the communication parameters pertinent to our discussion. The invocation descriptor includes a transaction identifier, addressing information, and the time fence field. The transaction identifier is filled in by the communication subsystem when a request or accept primitive is issued; this corresponds to the *key* value which uniquely identifies the context, and thus the association, for this communication. The addressing information is to allow ARTS to direct this communication to the proper thread and invoke the proper operation; XTP has a rich addressing facility. The time fence value is discussed below.

The message descriptor includes information about the message. This structure is shown as it is given in [TOKU90].

```
typedef struct MSGD {
  MSG_TYPE  m_type;     /* message type */
  u_short   m_pri;      /* message priority */
  MSG_ADDR  m_addr;     /* message address */
  u_long    m_size;     /* message size */
  u_long    m_errorno;  /* invocation error code */
} MSGD;
```

The `m_type` field indicates whether the message is `LONG` or `SHORT`, where `LONG` messages require segmentation. This field is to direct the Communication Manager about packetizing the message into more than one MAC frame; since transport protocols can handle arbitrary message length, XTP does not need to know whether a message is `LONG` or `SHORT`. The `m_pri` field supplies the message's priority. The *sort* field in XTP may use this value, but the size of `m_pri` (an unsigned short word) indicates that its value is intended for the priority field of a MAC frame. There is a size mismatch; XTP's *sort* can be more expressive. The `m_addr` and `m_size` fields give the message's buffer address and size respectively; this information is necessary to XTP as well.

Time Fence

Each operation within a thread has associated with it a time fence, or an expected time of completion. This time fence is given by a including a `within` *time* `except` construct following the declaration of the operation. This *time* is the deadline by which the operation must be completed. Since communication primitives are operations, the time fence is included in the invocation descriptor and delivered to the remote thread. This time fence value can be delivered with the message as out-of-band data (the *btag* field), since it is an attribute of the message rather than part of it.

Communication Manager

XTP uses priority queues whenever contention for processing within a processing node occurs. This approach is similar to the worker threads used by the Communication Manager. While a worker thread can be preempted at any point in time so that a more important worker thread may be processed, XTP does not preempt the processing of a packet until a “clean point” in the processing is reached. Since packets are bounded in size, the period of priority inversion is bounded. Furthermore, XTP’s design is highly parallelizable, so the need to preempt processing is not as acute.

Since the Communication Manager provides similar message processing functionality as a transport protocol, an XTP-based communication subsystem would probably not need a full-scale Communication Manager as described in [TOKU89].

6. Conclusions

The ARTS distributed real-time operating system uses timing information, especially period and worst-case execution time, to schedule its tasks. It attempts to guarantee the hard real-time tasks first, then reduce the response time on the soft real-time tasks. ARTS claims to take the same approach for communications; however, the period and worst-case execution times are not provided to the communication subsystem. What is provided are the priority and time fence values.

Communication within ARTS follows a Request-Accept-Reply paradigm common to transaction and remote procedure call systems. Currently, MAC layer data delivery services are used; segmentation of a message into packets, sequencing, and reassembling these packets into the original message are all done by a Communication Manager, which serves as the preemptable message processor.

XTP provides transport layer functionality, combined with high performance design goals and algorithms. XTP offers efficient single-packet association establishment, and easily provides a transaction paradigm. Routing services are included within XTP since XTP is a transfer layer protocol. XTP does not provide message latency guarantees, nor does it schedule messages beyond the use of a priority based scheduling scheme. Still, XTP provides functionality and performance that would benefit ARTS, and could represent an improvement over the current MAC-based functionality.

Below we list the benefits and drawbacks to putting ARTS on XTP. First we discuss XTP over ARTS, focusing on how XTP provides improved functionality and how XTP could be improved to satisfy the real-time needs of ARTS. Then we discuss ARTS over XTP, focusing on how ARTS could use the functionality of XTP.

XTP under ARTS

- XTP provides full duplex communication for arbitrarily long messages over an arbitrary internetwork topology
- XTP provides user controlled degrees of reliability, especially useful when providing full reliability may cause timing constraints to be violated.
- XTP provides out-of-band data delivery, perhaps useful for the time fence value.
- XTP provides a 32-bit wide priority (*sort*) field. The value is used to order processing of packets at every processing point, including delivery to the destination user. The *sort* field is wide enough to be a timestamp, such as a deadline.
- XTP can not bound latency.
- XTP provides only priority scheduling, while real-time systems require scheduling based on timing and criticalness constraints. The *sort* field may not be powerful enough to provide ordering based on these two properties.

ARTS over XTP

- ARTS requires a Request-Accept-Rely paradigm; XTP is well suited for this.

- ARTS provides a message priority and a time fence value, which may be given to XTP as the *sort* value and the contents of the *btag* field, respectively. XTP delivers both to the destination user.
- ARTS does not currently use internetworking; XTP provides this capability.
- ARTS could make use of the multicast facility. Currently there are no primitives which require multicast, though this could be more from lack of facility than lack of need.
- An XTP based subsystem could replace much of the Communication Manager.

7. References

[ANSI86] American National Standards Institute, “FDDI Token Ring Media Access Control Standard”, *Draft proposed Standard X3T9.5/83-16, Rev. 10*, February 1986.

[CHER88] Cheriton, D. R., VMTP: Versatile Message Transaction Protocol, Protocol Specification Preliminary Version 0.7, Computer Science Department, Stanford University, February 22, 1988.

[CHES87] Chesson, G., “The Protocol Engine Project”, *UNIX Review*, Vol. 5, No. 9, (September 1987).

[CHES88] 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).

[COHN88a] Cohn, M. D., “A Proposed Local Area Network for Next-Generation Avionic Systems”, P*roceedings of NAECON*, Dayton, Ohio, (May 23, 1988).

[COHN88b] 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).

[DARPA81] Postel, J. ed., “Transmission Control Protocol — DARPA Internet Program Protocol Specificating”, *RFC 793, USC/Information Sciences Institute*, (September 1981).

[FERR90] Ferrari, D., “Client Requirements for Real-Time Communication Services”, Technical Report Technical Report-90-007, International Computer Science Institute, March 6, 1990.

[IEEE85a] 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.

[IEEE85b] Institute of Electrical and Electronics Engineers, “IEEE Standard 802.4 Token-Passing Bus Access Method and Physical Layer Specifications”, 1985.

[IEEE85c] Institute of Electrical and Electronics Engineers, “IEEE Standard 802.5 Token Ring Access Method and Physical Layer Specifications”, 1985.

[ISO7498] International Organization for Standardization, “Information Processing Systems - Open Systems Interconnection - Basic Reference Model”, *Draft International Standard 7498*, October 1984.

[ISO8073] International Organization for Standardization, “Information Processing Systems - Open Systems Interconnection - Transport Protocol Specification”, *Draft International Standard 8073*, July 1986.

[LELA85a] 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.

[LELA85b] 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).

[LEHO87] Lehoczky, J. P., Sha, L. and Strosnider, J. K., “Enhanced Aperiodic Responsiveness in Hard Real-Time Environments”, *Proceedings of the 1987 IEEE Real-Time Systems Symposium*, San Jose, California, pp. 261-270, (December 1-3, 1987).

[MARL89] Marlow, D. T., “Requirements for a High Performance Transport Protocol for Use on Naval Platforms - Revision 1”, Working Papers, NSWC, (7/23/89).

[NOSC91] Naval Ocean Systems Center, “Military Handbook: Survivable Adaptable Fiber Optic Embedded Network II - MIL-HDBK-0036 (Draft)”, January 15, 1991.

[PEI89] Protocol Engines, Inc., “XTP Protocol Definition, Rev 3.5”, PEI 90-120, September 1989.

[SAE87] Society of Automotive Engineers, “SAE AS4074.2 High Speed Ring Bus, Final draft Standard”, June 1987.

[STRA89] 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.

[TOKU89] Tokuda, H., Mercer, C. W. and Ishikawa, Y., “The ARTS Distributed Real-Time Kernel and its Toolset”, Report, 1989.

[TOKU90] Tokuda, H., Mercer, C. W., Ishikawa, Y., *et al.*, “ARTS System Reference Manual”, Release 1.0, *CMU-CS-DRAFT*, The ART Project, Carnegie Mellon University, (October 22, 1990).

[WEAV89] 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).