Design and Analysis of a Transport Layer Reliable Multicast

A Thesis

Presented to

the Faculty of the School of Engineering and Applied Science

University of Virginia

In Partial Fulfillment

of the Requirements for the Degree

Master of Science (Computer Science)

by

Bert J. Dempsey

January 1991

ABSTRACT

Multicasting denotes a facility in a communications system for providing efficient delivery from a message's source to some well-defined set of locations using a single logical address. While modern network hardware supports multidestination delivery, first generation Transport Layer[1] protocols (e.g. the DoD Transmission Control Protocol (TCP) [14] and ISO TP-4 [54]) did not anticipate the changes over the past decade in underlying network hardware, transmission speeds, and communication patterns that have enabled and driven the interest in *reliable multicast*. Much recent research has focused on integrating the underlying hardware multicast capability with the reliable services of Transport Layer protocols.

This thesis explores approaches and solutions from the literature to the communication issues surrounding the design of a reliable multicast mechanism. Four experimental Transport Layer protocols that incorporate some form of reliable multicast are examined. The need for *multicast group management* services to augment Transport Layer multicast algorithms is identified, and such services are incorporated into a reliable multicast service designed as a part of the University of Virginia Computer Networks Laboratory software implementation of the Xpress Transfer Protocol. Performance measurements for the resulting reliable multicast facility are presented using a single-segment network.

[1] The Transport Layer is layer four in the International Standards Organization Open Systems Interconnect (ISO OSI) Reference Model ([34]).

ACKNOWLEDGEMENTS

I would like to express heartfelt thanks to my advisor, Dr. Alfred C. Weaver, for his guidance and patience in nurturing this work. Special thanks go to my colleagues in the Computer Networks Laboratory: especially Alex Colvin and Robert Simoncic, who contributed greatly to my understanding of XTP, and John Fenton who brought the Multidriver to life, among other things. I would also like to thank Dave Marlow of the Naval Surface Warfare Center and Larry Green of Protocol Engines, Inc. for their roles in funding this effort. Finally, I would like to thank my wife, Molly, for her understanding and support throughout all.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1

Overview of Multicast Issues

1.1. Introduction

Many distributed applications require efficient, reliable communication between a set of distributed processing entities, or a *process group*. Existing point-to-point protocols force the use of multiple unicast transmissions for group communications. These protocols are ill-suited for multi-party conversations in two fundamental ways. First, they are not designed to take advantage of underlying selective broadcast hardware support available on most modern networks. Second, since failure modes are more complex, the notion of a reliable transfer changes radically under a multi-party communication model, requiring functionality not present in existing point-to-point protocols. In particular, first generation Transport Layer protocols (e.g. the DoD Transmission Control Protocol (TCP) ([14]) and ISO TP-4 ([54])) did not anticipate the changes over the past decade in underlying network hardware, transmission speeds, and communication patterns that have enabled and driven the interest in *reliable multicast*. Much recent research has focused on integrating the underlying hardware multicast capability with the reliable services of peer protocols in the higher layers of the ISO OSI Reference Model. A *reliable multicast facility* is a communication protocol that provides distributed applications with reliable message delivery to a well-defined set of destinations using a single logical address and provides support for *group management*.

Multicasting frames at the Data Link Layer is supported in virtually all standard Media Access Control (MAC) protocols. Local Area Networks (LANs) conforming to the IEEE 802 standards ([31-33]), Ethernet ([22]), and the ANSI Fiber Distributed Data Interface (FDDI) standard ([3]) propagate frames such that all nodes on a frame's originating segment have the opportunity to capture it. Host interfaces to the network support hardware filtering on group addresses, and, as interest in multicast has risen, hardware for efficient address filtering has become increasingly sophisticated. Thus, each Link Layer frame can be delivered, within the constraints of the filtering interface, to exactly the set of destination hosts, or *host group*, for which the frame is intended.

Proposals have been made to extend Network Layer protocols, in particular the Internet Protocol ([14]), so that host groups can span networks. These efforts to provide a multicast capability for datagram packets traveling over wide-area networks (WANs) focus on host group management and efficient utilization of routing information ([8] [1]). There also exists a body of literature on routing multidestination packets over point-to-point links ([18][17][56]).

Considerable complexity arises in translating a machine-level multicasting capability into a reliable multicast facility. At the Transport Layer, the layer traditionally associated with a reliable messaging service, a multicast originator transmits messages using a single network address to a set of endpoints (*contexts*), the *multicast group*. As with unicasting, a Transport Layer multicast consists of two parts: a binding of a message's address to some set of receiver entities and a delivery mechanism to

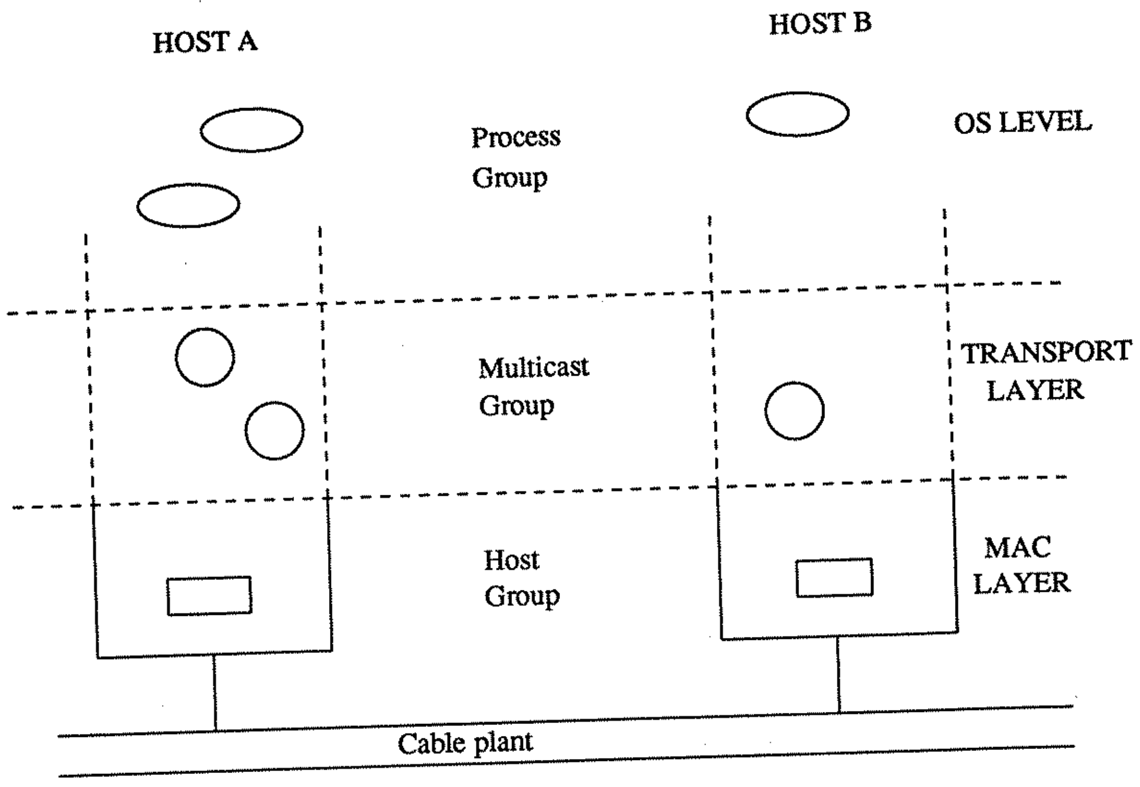


Figure 1.1 — Multicasting Terminology

deliver the message to every receiver entity to which its address binds ([8]). The functionality to perform the latter component must come from (next generation) Transport Layer protocols. As for the former, multicast addresses dynamically bind a logical set of communication endpoints to the physical set of endpoints currently listening on the multicast address. This distributed, run-time binding is both powerful and the source of much complexity since some ancillary mechanism outside the data transfer protocol must in the general case manage shifting group membership. This *group management* aspect of the multicast problem has no unicast analogue and requires a management entity that is properly located above (or beside) the Transport Layer.

Group management functionality includes handling reliability semantics relating to dynamic group membership.

1.2. Environments for Multicast

Three different network environments for multicasting may be identified: multicast over a *wide-area network* (WAN), a *multi-segment* environment, and a *single-segment* environment. Multi-segment environment refers to an extended LAN, i.e. one or more LANs connected by Network Layer relay nodes into a single addressing domain. A single-segment environment denotes a network consisting of a single LAN with links, if any, consisting of Data Link Layer bridges.

In a single-segment environment a multicast facility has no routing considerations and can expect (with high probability) nearly simultaneous delivery at the receivers. The high bandwidth and low latency of single-segments allow multicast conversations that are traffic-intensive, such as those in which receivers multicast their control information to the entire group as well as (or including) the transmitter. With the introduction of routing through Network Layer relays, the delivery characteristics experienced by receivers are much less uniform, and consideration must be given to the natural bottleneck at the router in any traffic-intensive scheme. Multicast across WANs introduces the possibility of having point-to-point links in the delivery path, which implies a different routing problem from that for multi-segment environments.

1.3. Multicast Applications

The need for multicasting arises naturally in a number of existing and emerging applications: resource location in a LAN ([2]), distributed databases ([6] [7]), industry process control ([38]), support for distributed operating system services ([37] [9]), replicated procedure calls ([15]), support for real-time command-and-control platforms ([41]), and collaborative development systems ([39]). One taxonomy of multicast applications classifies the behavior of the process groups as either *deterministic* or *nondeterministic* ([40]).

Deterministic process groups require strong data and behavioral consistency between their members. They use peer-to-peer communication, i.e., only members of the group send messages to the group. Examples include parallel processing entities sharing partial results and distribution of status information and coordination among components in automated control programs ([49]). Nondeterministic groups typically do not require the transmitter to be a member of the group. The prevailing model is of a client talking to a functional group of servers. Emphasis is placed on transparent group communication. Examples include resource location, replicated procedure calls, and most applications involving group querying and reporting.

One-to-many communication has inherent efficiencies when compared with equivalent service using multiple unicasts. Multicasting allows the source to generate only a single copy of the data, rather than one copy per receiver. Receivers process the distribution concurrently. If connection-oriented service is desired, a single one-to-many connection will most likely be faster and less costly to set up than multiple one-to-one connections. Multicasting thus speeds delivery and saves processing cycles at the source node, bandwidth, and remote host resources.

In shipboard or ground-based command and control environments, for example, signal processing techniques are applied to raw data from sensors and the processed data distributed across high-performance networks to display workstations for human operators ([13]). In [41] a scenario depicting the needs of future Navy platforms, specifically a Tactical Console Display subsystem, is discussed in detail. Twenty display workstations receive multiple data streams, one being a periodic update of the ship's primary track file in which various types of sensor data have been merged. A multicast capability is required to support rapid multidestination distribution of these graphics images, which range from one to ten Megabytes. The real-time constraints present in this environment make multicasting crucial since time does not permit a series of unicasts.

Multicasting offers fundamental benefits besides efficiency. Multicast addressing serves as a run-time binding mechanism for associating a group identifier based on a logical grouping of processes with the actual physical servers. A diskless workstation, for instance, may use, instead of a hard-wired unicast address, a multicast address for the group of boot servers ([8]). The number and location of the servers are unknown at the workstation and possibly change with time. More generally, this de-coupling of logical addresses and physical resources supports distributed data and resources through group querying and reporting.

This functionality will be useful, for example, in achieving substantial increases in network connectivity. A proposal being studied by the National Science Foundation for a National Collaboratory foresees the need for a very rich interconnection between multi-disciplinary scientists in order to accelerate the pace and quality of research projects such as mapping the human genome and global change ([57]). In the realm of application development tools, plans are now underway to move up the next step from distributed software development to *collaborative development* in which a number of contractors spread over a wide area will interact daily in the concurrent planning and developing of large software projects. This new software development environment will require multicast in at least three ways. First, there must be rapid file sharing among a number of physically dispersed sites. Second, the substantial increase in the total number of nodes on which project resources will reside will have a dramatic impact on Directory Services. In particular, the need for inquiries to distributed name and route servers will rise. Thirdly, collaborative development will require on-line electronic conferencing and electronic mail distribution lists to which interested parties can subscribe. Both of these applications are most naturally supported by a multicast mechanism. Existing projects represent first steps toward designing powerful distributed systems that provide the full range of services required for collaborative development.

Reliable one-to-many communication also opens up the possibility of synchronizing distributed processes without incurring the network-wide processing overhead and security problems inherent to broadcasting. If the current work on global time within a network proves successful, this property of a multicast may become especially valuable.

1.4. Organization

The provision of a general purpose reliable multicast facility involves functionality at several layers of the ISO OSI stack. Chapter 2 deals with the Data Link Layer at which group addressing and, in large LANs, routing multicast frames in bridges are issues ([18]). Chapter 2 also covers the Network Layer where the central problem is providing for the efficient routing of multidestination packets and maintenance of information on host groups. Chapter 3 examines the Transport Layer issues raised by one-to-many Transport Layer connections, including the development of control algorithms and the efficient collection and coalescing of control information at the multicast originator. Four Transport Layer protocols that address reliable multicasting are explored in detail. Chapter 4 discusses the need for a group management protocol to handle group membership requirements tied to reliable delivery between multipeers at the Transport Layer. A service interface for a group management facility to complement the Xpress Transfer Protocol's reliable multicast algorithm is proposed. Chapter 5 details the incorporation of group management functionality into the Multidriver, a reliable multicast service built into the implementation of XTP by the University of Virginia Computer Networks Laboratory. Performance measurements of the Multidriver and its extension are presented for a single-segment testbed network. Chapter 6 summarizes the conclusions from this work and notes the interesting possibilities for further work in this area.

Chapter 2

Multicast Issues at the Data Link and Network Layers

Several underlying mechanisms are necessary to support the efficient delivery of Transport Layer multicast messages. At the MAC sublayer, group addressing addressing and packet filtering hardware are widely available for sending multidestination frames. Proposals have been made to enhance routing algorithms for both MAC sublayer bridges and Network Layer routers to handle multidestination delivery.

2.1. MAC Sublayer Addressing and Packet Filtering

At the MAC sublayer, multicasting frames requires the capability of binding a frame's destination address to multiple hosts. Standard MAC protocols support this. The 10 Mbit/s Ethernet ([22]) reserves the two most significant bits to indicate if the address is a group address and whether the address is locally or globally administered. The remaining 46 bits are available to create 2^{46} unique group addresses. Similarly, the IEEE 802 MAC protocols for 802.4 Token Bus ([32]) and 802.5 Token Ring ([33]) as well as the addressing scheme for the ANSI FDDI standard ([3]) support a wide group address space.

In addition to the ability to create MAC sublayer group addresses, however, multicasting MAC frames relies on a host being able to recognize which multicast packets are intended for it. Ideally this *packet filtering* should be done entirely in the network interface hardware since doing it in software is orders of magnitude slower. The advantage of having large group addressing spaces is mitigated by the fact that current network interfaces cannot filter for more than a small number of group addresses, though hardware designers are paying increasing attention to this problem.

Consider, for example, the group address support provided by the TMS380 chipset, a popular token ring interface developed by Texas Instruments for the IBM 4 Mbit/s token ring architecture ([55]). The adapter board associates three addresses with a node at initialization: the ring station address, a group address, and a functional address. The ring interface also copies any frame with the destination address representing an all-stations broadcast. There are two bit patterns that represent an all-stations broadcast.

The token ring MAC address fields are 48 bits long (Figure 2.1). The high bit signifies a group address; the next-to-highest bit signifies whether the address is locally administered or universally administered. A ring station address uses the lower 46 bits. The group address format has a fixed bit pattern in the upper 17 bits and the group address in the lower 31 bits. Functional addresses also have a fixed bit pattern in the upper 17 bits and 31 bits of system-supplied address. When the destination address is a functional address, the station matches its functional addressing mask against the functional address. If any bit position is set in both, the station copies the frame from the network. Thus, functional addresses are encoded in a bit-significant manner, and any station may filter for any or all of the 31 functional groups. Five functional groups have already been designated by the IBM token ring architecture for special purposes (e.g. use by a bridge, network manager, or active monitor). A multicast facility has approximately 27 distinct group addressing filters — the 26 unclaimed functional addresses and the one

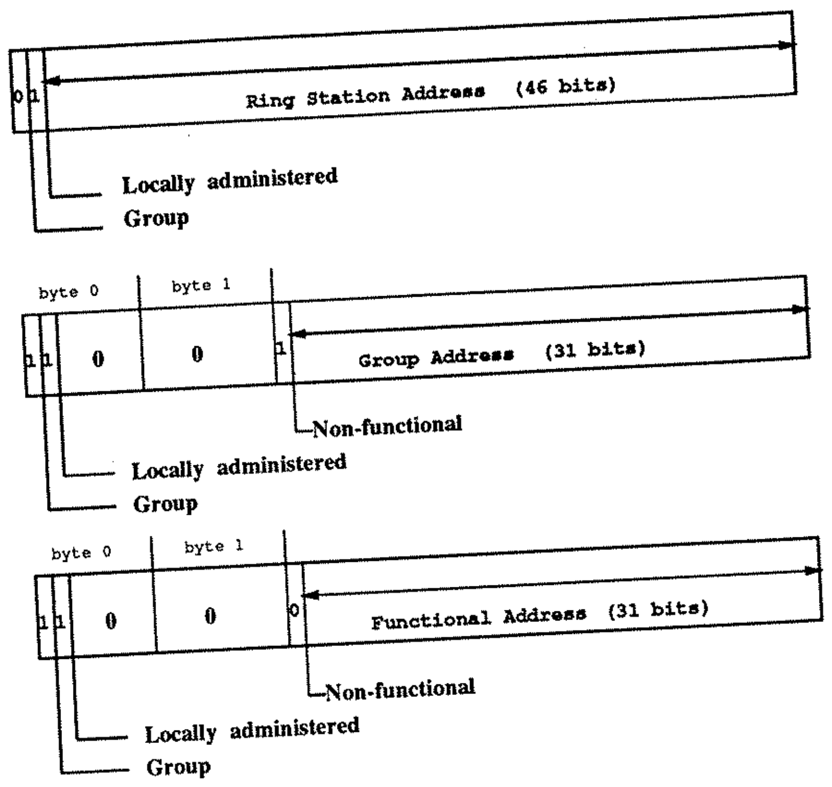


Figure 2.1 — 802.5 Token Ring Addressing

group address— supported in hardware at each node at any one time.

2.2. Routing

A LAN is constrained by limits on the number of stations, maximum distance between any pair of stations, and maximum traffic loads. Thus multiple LAN segments are often needed for a single community of network users. These multiple LANs are connected by intermediaries known in the ISO terminology as *relays*, and a relay may be present at any layer of the ISO OSI Reference Model. If the relay shares a common layer *n* protocol with other systems, but does not participate in a layer *n*+1 protocol in the process of relaying information, it is known in the ISO terminology as a *layer n relay*. Common terminology denotes a Physical Layer relay as a *repeater*, a Data Link Layer relay as a *bridge*, a Network Layer relay as a *router*, and any higher layer relay as a *gateway*. While this terminology is common, it is not used universally and one should be aware that the term *gateway* is sometimes used in the literature to describe a relay at any layer ([46]).

2.2.1. Network Layer

Routing techniques for multicast in store-and-forward networks have been examined in a number of contexts. Historically, these multicast routing techniques were first examined for point-to-point networks ([56]). Most strategies are built around spanning trees, a natural solution to the problem of taking an arbitrary topology and producing an edge set in which there exists exactly one path between any pair of nodes, i.e., eliminating cycles.

Networks in which, instead of point-to-point, multiple-access links connect routers, or *bus-based networks*, have different characteristics and routing criteria than point-to-point WANs. Cost is an important consideration in general for WANs, but plays no role in routing through a bus-based network, where packets do not incur tariffs. An excellent summary of techniques for multidestination routing in this environment appears in [25]. Recent work on this problem also appears in [42], though the authors point out that their algorithms for *multicast trees* are more appropriate for multiprocessors and multichannel LANs than for interconnected LANs or MANs since the algorithms depend on maintaining complete knowledge of the network topology at all network nodes.

The Internet Group Management Protocol (IGMP) was recently defined to extend the network layer Internet Protocol (IP) so that IP datagrams can be delivered to a host group instead of a single host ([19]). IGMP allocates a certain portion of the IP address space for host groups and addresses the mapping of IP addresses to MAC group addresses. The protocol mandates the existence of *multicast routers*, which may or may not be distinct from conventional IP routing nodes on the network, for routing IP datagrams addressed to a host group. The IGMP service interface allows higher layer users to join and leave host groups. Multicast routers learns of the presence of group members on an attached segment via periodic reports by member hosts. Membership reports are transmitted on the MAC multicast address so that hosts in the same group overhear each others' transmissions and avoid redundant reporting. For communication between multicast routers Deering ([18]) has proposed extensions to two standard routing algorithms for Network Layer routers—distance-vector routing and link-state routing—using in the former case refinements to *reverse path forwarding* ([17]). IGMP should soon begin appearing in commercial products.

2.2.2. Data Link Layer

The IEEE 802 protocols specify two sublayers within the ISO Data Link Layer, the Medium Access Control sublayer, located next to the Physical Layer, and the Logical Link Control (LLC) sublayer above. As intended by the 802 Committee, the term *bridge* refers not just to a Data Link Layer relay, but more specifically to a relay operating below the MAC sublayer service boundary within the Data Link Layer. This definition ensures that the relay will operate independently of all LLC and higher layer protocols. Bridges are store-and-forward routing switches that attach to two or more electrically independent cabling LAN segments. Hence a frame arrives on one of the cable segments, the *incoming link*, and is forwarded onto one or more *outgoing links*. A *bridged LAN* refers to a LAN in which all relays are bridges (or repeaters).

Two routing algorithms for bridged LANs have been endorsed by the IEEE 802 Standards Committee: the IEEE 802.1 Transparent Spanning Tree (TST) Scheme and the IEEE 802.5 Source Routing Scheme. *Transparent bridges*, bridges as defined by the IEEE 802.1 Medium Access Control Bridge Standard ([29]), provide transparency in the sense that end nodes do not participate in routing decisions. Instead TST bridges use a distributed algorithm to transform the arbitrary mesh topology of the given network into a single, acyclic spanning tree through which frames are forwarded. These bridges maintain a forwarding database of the location of nodes as determined through examination of the source addresses in frames. Topology changes are detected by intra-bridge communication, and a new spanning tree determined. The bridges self-configure upon initialization and even recover if misconfigured by human installers.

Source routing is based on including the route to the destination node(s) in a variable length field of the frame. Under this scheme, a bridge performs string matching on the routing field to determine to what links, if any, this bridge should forward the frame. Source routing has a number of advantages. Perhaps most important of all, source routing bridges are relatively unaffected as the size of the network grows and transmission speeds increase unlike TST bridges, which are tied to address-table maintenance and look-up. A major limitation to source routing is that a key element of dynamic route discovery by the source host consists of broadcasting frames throughout the network to explore all possible paths to the target ([23]). Consequently, its use is more appropriate in bridged LANs of small diameter.

In a bridged LAN of small diameter, multicast packets are simply broadcast to all segments. The abundance of switching resources and bandwidth compensate for the inefficiency of delivering packets to segments where no receivers exist. Filtering hardware offloads hosts in the task of discarding packets. Thus, added complexity in the bridge routing algorithms to achieve *scope-controlled* multicasting, a multicast that propagates a fixed "distance" from the originator instead of throughout the network, cannot be justified for small diameter networks. It follows that attention should be focused on the TST Scheme and not the Source Routing Scheme.

In [18] the authors propose extensions to the TST Routing Scheme to accommodate efficient multicast for large bridged LANs (on the order of 10 segments). The scheme augments routing tables to handle multicast addresses and dictates that the members of the multicast host group, G, issue periodic *membership report* packets by which bridges learn the links on which to forward packets with destination G. In this way bridges learn the paths for multicast packets and confine multicasts to portions of the network where members of the destination group reside. The overhead of sending membership reports in order that bridges can learn about the location of group members is shown to be very manageable. The primary drawback to this proposal is the loss of transparency in the hosts. The authors argue that the appropriate functionality may appear in future LAN interfaces and can in any case be provided by modifications to LAN device drivers, but for current systems such modification may not be deemed justifiable.

In [53] the authors develop algorithms that allow the use of (non-standard) bridges in extended LANs of arbitrary topology without confining the traffic to a single spanning tree. The scheme depends on decomposing the network graph into some number of spanning trees, numbering them, and then marking each packet as traveling on a single tree. The TST-bridge technique of building routing tables based on the source address of packets passing the bridge ([5]) can be preserved while traffic flows along multiple paths. Given the ability to perform such a multitree decomposition of the network, the authors go on to present a routing algorithm for efficient (i.e., *scope-controlled*) multicasting. The algorithm does depend on two-way communication to resolve a path so that hosts involved in the exchange must transmit at some guaranteed minimum rate in order for the bridges to retain the proper routing information.

The idea of using multiple spanning trees has a number of appealing characteristics. Like source routing, it allows dynamic load balancing, leading to better overall network performance, and in the case of a link failure, it enables a connection to switch very quickly to another route. Unlike source routing, all the preparation cost of determining and numbering a set of spanning trees (i.e. a set of well-known routes cached at each node) can be confined to network initialization time. Addressed-based table look-up suffers the same drawbacks as the TST Routing Scheme, only the address tables are even larger with multiple tree forwarding. And, of course, an implementation of this scheme would require special purpose hardware bridges, which may be expensive and risks interoperability problems with existing networks based on international standards.

Chapter 3

Transport Layer Multicast

Traditional Transport Layer (unicast) service shields higher layers from the details of the underlying unreliable network, including transparent recovery from lost or duplicated data. Since packets can be lost, the receiving context in most point-to-point Transport Layer protocols sends control packets back to the sender. These packets typically include an acknowledgement of received data (error control) and an indication of the availability of buffers for more data (flow control). In order to recover from lost control packets, the sender usually employs a timer. If the timer expires before the arrival of an expected control packet, the control packet is assumed lost, and the sender takes actions accordingly, e.g. requesting the receiver to issue another control packet.

The presence of multiple receiving contexts — reliable Transport Layer multicast — complicates this scenario. First and foremost there are group membership questions. A unicast address binds to a single, unique endpoint within the network. If that endpoint does not exist at connection set-up or fails during a data transfer, then the transmitter easily detects the failure since no control information arrives. With the dynamic binding of multicast addresses, a connection involving some but not all members of the multicast group is possible. Higher level mechanisms must ensure that, in any given exchange, group membership is 'correct'. Even if membership is 'correct', a Transport Layer transmitter may not have a separate control channel for each receiver and hence can not know when all multicast group members have reported their status.

The technique of making the control channel reliable by timing out lost control packets encounters problems when extended to the multiple receiver case. First, the timer must be based on the maximum of a set (possibly of unknown cardinality) of roundtrip times. Second, if a time-out occurs and a control packet from each receiver has not been received, then the protocol may act on the partial report from the receiver group and risk making a wrong decision that degrades the efficiency of the transfer or, worse, loses data due to the premature release of a transmit buffer. Alternatively, the receivers not responding can be offered another chance to respond. This approach, however, leads to the problem of how to contact these silent receivers. Two possibilities exist: (1) the sender initiates a new response from all receivers, which may be expensive in terms of network resources and may well result in another partial report from the set; or, (2) the sender attempts the potentially prohibitively slow action of unicasting requests for control packets to each silent receiver, assuming that all receivers are known individually. A reliable Transport Layer multicast mechanism must specify one-to-many (flow, rate, and error) control algorithms that are robust and efficient in the face of partial updates.

For many-to-one data flow within a LAN (e.g. collecting acknowledgements from the receiver set), the phenomenon of *network implosion* must be addressed. Under any transmitter-driven control scheme the set of multicast receivers will tend to synchronize the sending of their control packets. Synchronized transmission can result in bursts of traffic on the network and the inability of the multicast source's network interface to capture frames arriving back-to-back.

Even if all receivers send control information, the multicast transmitter must collate the multiple status reports into directives that drive the multicast transfer. When the sender determines that data has indeed been lost in transit to some subset of the receivers, for example, the data must be retransmitted. If retransmissions are multicast, when a single receiver or a small number of receivers causes retransmission of a data packet, there is much work lost in resending data to the receivers who have already successfully received it. If retransmissions are unicast, the sender may have to frame and send a large number of copies of the same data.

3.1. Four Transport Layer Reliable Multicast Mechanisms

Reliable Transport Layer multicast mechanisms must first ensure effective collection of control information from multiple receivers and secondly specify robust one-to-many control algorithms. Below we examine four Transport Layer protocols that address reliable multicasting. These protocols emerge from different design philosophies and assumptions about use, performance, and environment. Perhaps the most important difference to note in comparing their approaches to reliable multicast are the assumptions about group management support from higher layer protocols. Two of the four assume that the transmitting context has been supplied with an explicit list of the receivers at the beginning of the data transfer; the other two do not posit any group management entity.

3.1.1. CP

The Transport Layer reliable multicast protocol proposed in [16], called here CP, represents a straightforward, but detailed attempt to handle reliable multicasting by having the multicast sender manage separate transmit windows for each receiver. The protocol assumes that process group membership is managed by some mechanism that allows the Transport Layer user to state the group membership and lock onto it for the duration of each individual exchange. The transmitting multicast context therefore has a list of group members. CP supports a range of reliability requirements, gives explicit consideration in its design to the possibility of internet links of low-bandwidth and/or a point-to-point nature in the delivery path, and has two proposed service interfaces.

3.1.2. Versatile Message Transaction Protocol

The Versatile Message Transaction Protocol (VMTP) ([11]) is designed as a next generation protocol to accommodate communication strongly oriented toward request-response behavior and uses the transaction paradigm as the basis of all communication. Reliable multicast transactions are defined as transactions with group entities in which at least one response from the multicast group is received. Responses after the first one are buffered for the user and delivered if requested. Hence, messaging service reliability depends, beyond the initial response, on the reliability of user-level transactions. The V Distributed Operating System ([9]), to which the development of VMTP has been closely coupled, defines a service interface that includes process group management primitives. VMTP itself has an integrated management facility that handles creating, modifying, and querying for group entities (multicast group identifiers).

3.1.3. Xpress Transfer Protocol

The Xpress Transfer Protocol ([12][47]) (XTP) is a lightweight transfer layer (the transfer layer being defined as the Transport and Network Layers merged) protocol being developed by a group of researchers and developers coordinated by Protocol Engines, Inc. It is designed to provide the end-to-end data transmission rates demanded in high speed networks such as FDDI and the gigabit/sec wide area networks without compromising reliability and functionality, including in particular, support for reliable multicast. XTP intends to accomplish its goals through streamlining the protocol, combining the Transport and Network layers, and utilizing the increased speed and parallelization possible with a VLSI implementation ([51]). XTP defines a reliable multicast mechanism such that a transmitting context, knowing only the group address, can perform a flow, rate, and error-controlled one-to-many message delivery. Like VMTP and unlike CP, the mechanism described has been carefully designed so that reliable multicast imposes a minimum of overhead on unicast protocol processing. The reliability guarantee is fragile in the sense that transmit buffers are released based on estimations of the maximum roundtrip time between the sender and the receiver set.

3.1.4. NAPP

A mechanism based on Negative Acknowledgement with Periodic Polling (NAPP) ([50]) takes the novel approach of having background daemons at each receiver that assure progress and periodically send liveness messages to the source during a multicast distribution. The mechanism assumes that a one-to-many 'virtual circuit like connection' has already been established; thus, the sending context has explicit knowledge of the receiver set. Receivers multicast control information so that all group members overhear each other, and each control message that reaches the multicast source contains a report on all receivers' sliding windows. This relaying of control information reflects a fundamental assumption underlying the design of NAPP, namely that the failure of one receiver to receive a packet strongly suggests that others have missed the packet as well.

3.2. Flow Control

Flow control refers to the receiver's ability to throttle the source in order not to overrun the available buffer space on the receiving host. A multicast exchange's flow control must be governed by the minimum of the flow control parameters for all the receivers in the exchange. The alternative is to allow a situation in which some subset of receivers is deliberately overrun, a strategy that would normally be counterproductive.

Maintaining proper flow control parameters at the sending context is particularly important since hardware improvements have produced networks with vanishingly low bit-error rates, meaning the majority of errors on these networks will occur due to incorrect flow control. The control information collection strategy should ensure that the transmitter knows about or quickly learns the correct flow control parameters at connection set-up. If for any reason during a data exchange (i.e. early release of transmit buffers) the receiving group is pruned, the transmitting context should ideally recompute the new minimum flow control parameters since some slow receivers may have been dropped.

XTP, CP, VMTP, and NAPP base flow control on the most limited receiver in the receiving group. The success of their flow control algorithm therefore depends on the effectiveness of their control packet collection schemes. Unlike CP and NAPP, XTP and VMTP do not assume that the multicast transmitting context has explicit knowledge of the receiving group and therefore cannot know with absolute certainty whether all receivers have reported their flow control parameters or not.

3.3. Collection of Control Packets

3.3.1. Simple Reporting: CP

The CP protocol takes the simplest approach to structuring the flow of control information by having multicast receivers unicast their control packets to the source, which knows the receiving set and manages a control channel for each receiver. The strong reliability guarantees possible under this scheme result from (1) the maintenance of a control channel for each receiver and (2) the underlying assumption of a powerful group management support facility. This facility maintains a lock on the multicast group membership for the duration of an exchange and notifies the transmitter should a server leave the group abnormally. As for network implosion, the designers of CP acknowledge the problem, present some mathematical analysis of it, suggest some general approaches to dealing with it, and finally leave it to implementors of the protocol to solve.

3.3.2. User-Level Responses: VMTP

In the VMTP unicast, a transaction starts with a client issuing a request to a server entity. At the server, on-demand connection set-up creates a transaction record upon receipt of the request. It is expected that a response packet containing the user-level response data will usually be quickly generated at the server, and that this response packet will function as an acknowledgement to its associated request. Otherwise, based on a time-out, the client sends a demand for an explicit acknowledgement to the server, who responds immediately. In this way, the client is assured that the delay is due to server processing and not because the request was lost.

VMTP's multicast capability focuses on compatibility with unicast mechanisms. A multicast transaction follows the same sequence of events as described above for unicasts except that the client sends its request to a group entity. The VMTP sender sets its timer upon issuing a request. Receipt of the first response disables the timer, and thereafter the VMTP client awaits responses without taking any further action. If the timer expires before any response arrives, the VMTP client issues a demand for an immediate acknowledgement from the group of servers. No special mechanisms address network implosion, though user-level acknowledgements generally produce a much greater variance than control packets generated within the communications protocol itself, making implosion less likely.

VMTP's 1-reliable multicast primitive is tailored toward the protocol's target environment of rapid exchange of small amounts of data over a network with low error rates (e.g. remote procedure calls over LANs). (The protocol accommodates large requests and responses through *packet groups*.) For the common case of a single packet multicast request, the VMTP multicast provides a low-overhead service. The application-level transaction determines reliability beyond the initial response, which indicates a high probability that the members of the server group will see the request. For the *k*-reliable semantics of multicast introduced in the V System, this 1-reliable primitive appears to be adequate. The specification of VMTP ([10]) does not explicitly describe how to provide flow-, rate-, and error-controlled one-to-many delivery of multi-packet requests, though it could be argued that the existing protocol features are adequate to build such a service.

3.3.3. Adaptive Timers and Slotted Damping: XTP

XTP protocol processing for multicast uses a *bucket algorithm* at the sending context. As with unicast, the multicast transmitter sets a status request bit (SREQ) in an out-going packet in order to get receivers to issue a control (in XTP parlance, CNTL) packet. The time at which a request was issued as measured by the XTP *sync* counter, which is incremented by one each time a packet is transmitted from the context, is recorded at the transmitter. Each in-coming CNTL packet contains an *echo* value that reflects the value of *sync* as known at the receiver. (If the sender uses a CNTL packet to send a SREQ, this packet will contain the new *sync* value.) The *sync/echo* mechanism enables in-coming CNTL packets to be sorted into buckets by "age", i.e. *sync* value, and CNTL packets in the same bucket are coalesced by recording the minimum of the flow parameters, the byte-based sequence number (*rseq*) that indicates the highest consecutive sequence number received without error at the issuing receiver, and the maximum of the round trip time estimate. The *wait interval* for determining when to act on the oldest bucket is calculated by a smoothing procedure suggested by Van Jacobsen to the round trip time values obtained in CNTL packets. Unless transmit buffers are mistakenly released early, the XTP error control algorithm ensures that all correctly functioning receivers will eventually receive the multicast distribution.

In XTP a multicast receiver issues a control packet to the group address. For large groups the XTP specification document ([48]) suggests a heuristic, called *damping*, that lessens the number of superfluous control packets flowing to the transmitter and addresses network implosion. Damping requires that receivers, upon seeing another receiver's control packet, dequeue any control packets that they have which contain an *rseq* value greater than or equal to the overheard value. Damping does not presuppose any knowledge of group membership at either the sender or the receivers.

The primary disadvantage to the proposed damping mechanism results from its fragility due to timing considerations that may vary widely over disparate environments. It is not clear that a node, R, can receive CNTL packets and perform quickly enough the processing necessary to locate and dequeue R's own CNTL packet. These timing concerns may seriously jeopardize the robustness of the multicast mechanism. Accordingly, a second heuristic known as *slotting* is introduced in the specification document. Slotting refers to the introduction of a random back-off time before a receiver generates its CNTL packet. This technique allows time for CNTL packets to arrive that might damp the out-going CNTL packet.

While an enhancement to just damping alone, slotted damping, like damping, seems susceptible to environment-specific timing dependencies. The density, homogeneity, and number of receivers all will play a role in the effectiveness of the slotting technique. A group management scheme for XTP could provide hints about these group characteristics. The XTP specification document also notes that slotted damping is compatible with the bucket algorithm, but care must be taken to insure that slot delay is less than the lifetime of a bucket ([48]).

XTP multicast is currently focused on multicast over a single segment. In particular slotted damping may not prove effective in avoiding congestion problems at routers due to the multicasting of all XTP Protocol Data Units (PDUs). Modifications or alternatives to the slotted damping heuristics may be considered by the designers of XTP as multicasting matures. One approach would be the collection of control information through relaying information back to the transmitter via a control channel structure such as a tree or a ring. The multicast sending context, the receiving contexts acting independently, or group management could establish and maintain the relay route. If the structure is created at connection set-up time, long-lived connections are preferable since they amortize more efficiently than short-lived connections the cost of set-up.

Consider the following scenario. Each group will contain a small number, say four, of special receivers, called *collectors*, from which any sender to the group will receive all control packets for the transfer. When a processing entity joins a group, it is given a collector's address, to which the new group member will unicast its control packets. Collectors coalesce control packets and relay (unicast) them in a single composite control packet to the multicast sender. A collector needs only enough knowledge to set up its address filters correctly and some logic with which to coalesce control packets. From the multicast sender's viewpoint, the data transfer is considerably simplified. The sender establishes a connection with the group and receives a fixed number (here four) of control packets on each sender-generated request for control packets as well as error reports whenever errors occur ([21]).

This caching strategy offers many benefits. Four collectors would widen the bottleneck at the source host's network interface by a factor of four. The two-step unicasting of control packets avoids the potential for generating a large number of packet interrupts at participating hosts. This consequence is inherent in the schemes that have receivers multicast their control packets to the group (like XTP). Collectors perform a sort of localized suppression of control packets in coalescing packets. This feature would be useful in reducing traffic through routers for a group residing on multiple segments and in partitioning the problem of collecting control packets for large groups on a single-segment LAN. Moreover, the collector algorithm could possibly be designed to collapse to a simple scheme of receivers unicasting responses directly to the sender for small groups that do not need two-step control packet reporting. The delay of relaying packets, especially on single-segment LANs, the heavy processing duties of the collectors, and the overhead of managing the relay structure represent the primary drawbacks to this idea.

3.3.4. Polling: NAPP

NAPP and CP have similar design goals in the following sense. Both protocols emphasize a high degree of reliability in multicast data delivery at the expense of producing lightweight, fast protocols. (For XTP and VMTP, the trade-off is roughly the opposite.) In NAPP the multicast source transmits data and performs retransmissions based on control information from the receivers. Like XTP, NAPP uses multicast control packets so that receivers may monitor each other's state and thereby reduce the amount of control traffic. NAPP receivers, however, interact in a far more complex manner than the simple damping behavior found in XTP.

Receivers issue three packet types for control information: ACK, PACK, and SREJ. All three are multicast so that receivers overhear and monitor each other's state upon transmission of every control packet. All three contain a *state vector* reporting the highest in-sequence packet received at each of the receivers. The data source uses the in-flow of state vectors to decide when to slide forward its transmit window.

Some terminology is needed for the discussion of NAPP that follows. Let M be the maximum number of packets that can be outstanding, that is, pending to be acknowledged at any one time. Let V_i be the first in-sequence packet not received at receiver *i*. Finally, let W_i be the window of receiver *i* consisting of packets sequenced V_i, ..., $V_i + M - 1$. All timers are assumed to have a granularity of milliseconds.

A receiver issues a poll-cum-acknowledgement (PACK) every T_{pack} milliseconds. The PACK is numbered V_i (expressed here as PACK(V_i)) and serves to solicit (re)transmissions, if any, of packets in W_i and acknowledges the packets in the range $V_i - M$, ..., $V_i - 1$. A PACK is rescheduled for T_{pack} milliseconds later upon reception of a packet in W_i, upon transmission or receipt of an SREJ(m), m W_i, or upon receipt of a PACK(q), $q \geq V_i$. Thus, PACKs serve as sort of background daemons that are never actually transmitted as long as data continues to flow to the receiver.

An SREJ(m) packet is scheduled for transmission by a receiver as soon as message *m* is detected as being lost. However, any SREJ packet is transmitted at its scheduled transmission time only with probability P and otherwise rescheduled for T_{srej} milliseconds later. When a receiver, R, overhears another receiver's SREJ(m), if message *m* is known to be lost already, then its own SREJ(m) is rescheduled for some time later, presumably putting off SREJ(m) long enough that the overheard SREJ(m) will have gotten message *m* retransmitted in the meantime. Any scheduled SREJ(m) is dequeued upon reception of *m*. If message *m* has already been received at R, the overheard SREJ(m) is ignored. Otherwise, message *m* is now perceived to be lost, and a SREJ(m) is scheduled for later transmission. Furthermore, receiver R checks to see if any messages from V_i, ..., m - 1 are lost. Thus, the reception of SREJ(m) at the source serves to acknowledge (possibly redundantly) $V_i - M, ..., V_i - 1$.

The third component in the trio of control packet types is an ACK(p) packet. ACKs are positive acknowledgements that receivers issue upon receiving some number of packets in sequence. To ensure reliable delivery, upon transmitting an ACK, a receiver reschedules the transmission of the same ACK for T_{ack} milliseconds later. ACKs are not necessary for the correct working of the protocol, but they do speed up the process of conveying acknowledgement status and advancing the source's transmit window. ACK(V_i) acknowledges $V_i - M, ..., V_i - 1$ at the source. Also, receivers overhear other receivers' ACKs and use them to monitor status in ways similar to those outlined for PACKs and SREJs.

Though there are more aspects to NAPP, this description gives the flavor and the most important aspects of its operation. In an actual implementation of NAPP, much attention must be given to the mechanisms to determine the correct settings for its many timers; the paper describing NAPP ([50]) does not focus on these implementation details, but instead notes the relative lengths of timers, e.g., $T_{\text{pack}} > T_{\text{ack}}$. A primary drawback to NAPP's approach is the management of adaptive timers in the face of dynamic system parameters, changes in group size, and connections made by multicast sources of varying processing power. The defaults for the timers that drive the background daemons at each multicast group member may be inappropriate for a particular connection. Short transfers will suffer unpredictable delays and/or periods of temporary instability as timers adapt.

3.4. Error Control

Error recovery at the multicast source must address whether to unicast or multicast retransmissions and whether to selectively retransmit or use go-back-N. Go-back-N requires less processing to determine the exact data that was lost, but risks generating large amounts of data if multiple requests for the same data are processed at the multicast source. Multicasting retransmissions burdens up-to-date listeners with processing duplicate packets. Unicasting to the unsuccessful receivers forces the sender to frame and send multiple copies of the data and to know how to address individual group members.

XTP uses go-back-N multicast retransmission and relies on an implementation having a robust method for calculating of the proper processing checkpoints. VMTP does not specify multicast retransmission policy, though the logical choice seems to be selective multicast retransmission. NAPP provides for selective multicast retransmission. Multicasting the data follows from the NAPP designers' belief that the loss of data at one receiver strongly suggests loss of data by other set members. Selective retransmission makes sense given the sophisticated interaction between NAPP receivers, which reduces the possibility of retransmission requests overlapping. Finally, CP employs a more elaborate mechanism. In deference to the possibility of a multicast exchange over low bandwidth delivery paths, retransmissions are unicast to individual receivers unless the proportion of failed deliveries to group size is larger than some user-supplied threshold value. In the latter case, retransmissions are multicast to the entire group. The threshold value would be set based on the number and relative dispersion of the host group, lower for groups on LANs and higher for groups across internetworks.

Chapter 4

Multicast Group Management

4.1. Introduction

The presence of multiple receivers in a Transport Layer exchange introduces reliability issues related to communication with the 'correct' multicast set. As support for multicast at the Transport Layer is a relatively recent development, little attention has yet to be paid to designing auxiliary mechanisms for determining and controlling the membership and status of multicast groups. We believe that this *multicast group management* (MGM) functionality is properly handled outside of the Transport Layer itself (see Section 4.3), though the exact nature of a MGM facility will vary with the multicast primitives and service interface of the Transport Layer protocol that is being supported.

A body of literature exists on *atomic multicast protocols* ([6][49][26]), which are designed for distributed applications that require strong reliability and message ordering guarantees. For this class of applications, reliability is typically defined as atomic delivery — if one process receives the message, then all do — and consistent message ordering — all multicast messages addressed to the same group are delivered in the same order to all the process group members, even if the messages originate from different groups. Atomic multicast protocols are process-level strategies for maintaining these requirements in the presence of process and network failures. Techniques include passing a token among group members ([49]), having a rotating *primary receiver* ([7]), and two-phase sequencing protocols ([6]).

In contrast, by multicast group management, we mean a facility that would provide applications with a set of services much less powerful, but also much less expensive, than those of the higher layer atomic multicast protocols. MGM primitives are intended to enable the application to discover and control multicast group configuration and to express membership requirements precisely for a particular multicast exchange. At least within a client-server model for multicasting, MGM represents a general-purpose mechanism by which applications can tune the Transport Layer multicast service using application-specific knowledge. A properly designed MGM facility augments the multicast data transfer protocol with the goal of providing the Transport Layer user (at the transmitting node) with control over the configuration of the receiving set and programmatic reliability for each instance of group communication.

4.2. MGM Possibilities

We now look back to the protocols examined in Chapter 3 to see how they can be related to MGM.

CP and NAPP specify that the multicast sender have explicit knowledge of the multicast group. NAPP assumes that some separate mechanism establishes a "virtual circuit like connection" ([50]) between the sender and the receiver set. During this connection phase the sender builds a list of the receivers. Once the connection phase completes, the loss of a multicast receiver causes delivery to the group to fail. Hence process group members leaving the group after a connection has been formed should be reported to the multicast sender if the exchange is to be completed and we are to avoid rebuilding the connection. Though NAPP as defined does not address group management, the polling acknowledgment (PACK) mechanism at each receiver is well-suited to handle reporting membership changes.

CP specifies a transaction-based service interface in which the user can pass two parameters, a list of essential servers and a minimum number of servers, that together determine the required multicast group. The designers of CP posit the support of a group management facility that allows the multicast sender to lock onto the group membership for the duration of a single exchange. Specifically, a server which has begun responding to a request is prevented from leaving the group, and, if the server leaves abnormally, its departure is reported to CP.

VMTP and XTP define multicast primitives that do not depend on enumerating the receiver set. VMTP relies on the higher layer client to determine the success of the transaction based on the number of responding servers. A more powerful multicast service could be constructed by incorporating group management functionality into the management module specified to be co-resident with every VMTP host implementation. This module, which has a shared memory interface with its associated VMTP instance, appears to remote sites as a server and currently handles, for example, callbacks for authentication on secure requests.

XTP provides reliable one-to-many data delivery in the sense that control messages from the receiver set that arrive within a time window are processed ([48]). All XTP

PDUs, data and control, are sent to a group address. Control algorithms at the data source continue to function as long as at least one member of the receiving set is active. Lagging or failed receivers are dropped from the exchange without notifying the XTP user.

Under the XTP multicast algorithm, applications have no ability to enumerate the multicast set or to achieve reliability beyond probabilistic delivery based on time-outs. To some extent applications could handle group management directly using *concentration*, a technique whereby contexts for a set of in-coming data streams are spawned from a master context. An XTP implementation can be specialized for this behavior ([48]). A MGM facility offers primitives to handle group management tasks so that every application does not have to perform management from scratch. Also, XTP's target environments such as the 100 Mbit/s FDDI network can support on the order of 1000 nodes on a single segment. Multicast applications in these environments can be expected to involve host groups with as many as a few hundred hosts. With such large receiver sets, concentration becomes resource-intensive and clumsy.

4.3. Anatomy of MGM

Multicast group management is a service distributed across all hosts within the management domain. Though the service could be controlled by a central agent, robustness and efficiency argue for a local agent at each host, a *group management entity* (GME), that handles group management in a fully distributed fashion. These local agents offer two service interfaces: one to application processes and the other to the Transport Layer protocol.

Application processes will use the mechanisms available through MGM to determine and control multicast group configuration and enumeration. These application processes may be distributed applications themselves or process-level group management protocols implementing, for instance, distributed operating system services ([9]). MGM can provide group polling mechanisms and some low-level control over the network-visible distributed entities (multicast group members) associated with a process group. This functionality is required by many process-level group management schemes, may be difficult to achieve outside the communications protocol, and possibly can be more efficiently implemented within MGM. On the last point, for example, polling in which GMEs use the Transport Layer protocol may invite, based on knowledge about the set of MGM internal messages, the design of a streamlined many-to-one communication mechanism that could not be supported in the general Transport Layer interface.

Multicast group management functionality will allow the Transport Layer protocol to support reliable multicast services that guarantee membership management within a one-to-many data exchange. If the Transport Layer uses an explicit list of the individual receivers in an exchange, MGM can aid in list creation and management. If the Transport Layer multicast primitive does not explicitly track group members, then group management provides the polling service and handles the state information necessary to insure a user-specified membership for an exchange. Since the GME can interrogate the Transport Layer protocol for state variables, an exchange-specific census can include a checkpointing of receivers' progress, if the application is willing to incur the overhead of this polling. Group management services also may include the means to implement application-specific policies concerning whether members may drop into or out of the multicast connection. If the user-supplied membership requirement is violated due to shifting membership, then the Transport Layer protocol is notified and the connection released. This approach is consistent with regarding the required group membership as a Quality of Service parameter on which the multicast connection is contingent ([4]).

While the overhead of communicating and updating group state will not be worthwhile for all applications, we believe that the absence of the ability to explicitly track the members of a multicast group within an exchange represents a severe limitation for a significant number of potential multicast users. On the other hand, moving the membership-tracking functionality outside of the Transport Layer is essential with lightweight, next-generation Transport Layer protocols like VMTP and XTP. These protocols can not afford to have complex multicast primitives that would jeopardize their design goals, including, in the case of XTP, a VLSI implementation. In order to accommodate the growing number of distributed applications that will demand a wide range of multicast services, the enhancements of MGM are needed.

4.4. Placement of Group Management in the OSI Reference Model

Reliable multicasting has only just begun to move from research topic to working communications tool. The question of how the multipeer model will permeate the OSI Reference Model architecture is only now emerging in the ISO standardization committees ([44]). The direction that this work takes will determine the final placement, focus, and form of the entity or entities that will provide group management functionality. The placement of MGM within the OSI framework must consequently be considered an open question at this time.

Natural parallels exist between the role of MGM and two management entities defined in the OSI Model: (1) system management at the Application Layer ([24]) and (2) the Transport Layer Management Entity (LME) ([28] [45]). Embedding MGM within the LME of the Transport Layer is a "plausible scenario" ([45]) that fits well our definition of MGM. The interaction between a LME and the layer which it manages is implementation dependent and therefore no restrictions are placed on the nature of this interaction ([28]). The LME would encapsulate knowledge of the multicast contexts as an object to be managed. (A template for encapsulating Transport Layer objects to be managed has been defined by the ANSI X3T5 Committee ([24]).) Event notification based on shifting membership within a multicast group is a form of configuration and/or fault handling. As for the interface between a LME and an arbitrary application process, the process would submit Common Management Information Service (CMIS) ([35]) requests to the Application Layer, which are translated into Common Management Information Protocol (CMIP) ([36]) PDUs. These CMIP PDUs would then be communicated, possibly directly, to the LME, which would decode the CMIP PDUs and service the request ([28]).

4.5. XGM

In this section we propose a service interface for a MGM facility, XGM, that assumes XTP to be the Transport Layer protocol being supported.

4.5.1. XGM/Application Process Interface

census(group, &subgroups, [&member-list], &number, timeout)

partition(group, from-subgroups, to-subgroups)

merge(group, to-subgroup, from-subgroups)

state(group, [subgroups], [member-id], ...)

Multicast groups are identified by a multicast Network Layer group address used as the value of the **group** parameter. XTP Revision 3.5 ([48]) relies on IGMP ([19]) to define a mapping between the Class D IP multicast addresses used within XTP and 48-bit IEEE-compatible MAC group addresses. If an addressing scheme other than IP is used, a mapping to MAC group addresses must be specified. **Member-ids** represent a value that uniquely identifies a multicast group member (context) on a host, which is the role of XTP's *key* values (ignoring the most significant bit). Having no knowledge of how **member-ids** are generated, applications will interpret this value as an unsigned integer. Any individual multicast group member can thus be identified by a <MAC address, member-id> pair, which is the form of the (optional) **&member-list** in **census()**.

An application process uses the **census()** function to discover a multicast group's current membership. Knowledge of individual set members may be useful for group management purposes or, if the XTP service interface permits, for specifying the inclusion of individual group members or subsets in data transfers. For efficiency, **&member-list** is returned only when required. The **census()** primitive will always return the total number of members that respond to the poll (**&number**) and the union of all subgroup identifiers for the polled members (**&subgroups**) (discussed below).

XGM associates with each multicast member an n-bit vector, a *subgroup vector*. Each bit position of the subgroup vector represents a logical subgroup. A group member belongs to the k^{th} subgroup if the member's subgroup vector has the k^{th} bit position set. Each member must belong to at least one subgroup and may belong to any number up to n. While XGM assigns each multicast group member to a random subgroup on creation, the group management primitives collectively are designed to insure that application processes can exert complete control over the mapping of set members to subgroups.

This simple partitioning mechanism has a number of attractive properties. By default, given an initial assignment to a random subgroup, multicast set members are distributed uniformly over the range of subgroups. This arrangement contributes to efficient and effective group management for large groups. Each GME can take the union of all subgroups to which local group members belong and use this composite mask as a fine-grain software filter for determining whether to respond to a polling request from a remote GME. Multiple sequential polls or distributed polling can be used to enumerate very large groups while using a bounded number of XTP contexts at the concentrator.

Dynamic grouping of set members can take place under application control. Assignment of a common subgroup vector can isolate special group members such as routers or group members that are related in some application-controlled way. Unlike flat address spaces such as IP group addresses, subgroup vectors allow for hierarchical partitioning schemes. Moreover, by developing subgroup assignment conventions known and practiced by all processes in a distributed application, dynamic reassignment of subgroup members need only take place rarely, e.g., in response to failures.

If XTP multicast contexts were specialized to incorporate the filtering of FIRST packets based on subgroup bits carried in the packet, then users would have logical scope control when multicasting data. The most efficient filtering would link subgroup membership with MAC layer group addresses in order to allow filtering to take place at hosts' network adapters. A client in need of a single server from a set of identical servers would, for example, be able to restrict its request to less than the entire set of servers. Multicast services that reside at well-known addresses could use large numbers of identical servers without burdening the network with a correspondingly large number of responses to each service request.

In order that applications have the full flexibility of the subgroup mechanism available to them, XGM provides primitives to change the group partitions, e.g., **merge()** and **partition()**. These primitives propagate to the host group with datagram reliability. Reliable delivery is not meaningful since we do not posit any restriction on joining and leaving groups. In order to avoid difficulties with correct connection monitoring (see Section 4.5.2), a multicast group member involved in a connection can have its subgroup vector changed only if the change request comes from the GME at the data source of the connection.

In addition to the **census()** and subgroup vector management primitives, XGM should provide a service primitive by which the user can read and set variables associated with a multicast member, a subgroup, or a multicast group. One variable that must belong to the user's interface is the subgroup vector. An application should be able to read and write the value of its local multicast member's subgroup value. Group parameters could include a cached estimate of group size and default timer settings for establishing a connection with the group.

4.5.2. XGM/XTP Interface

connection_monitor(group, connection-id, membership)

release_monitor(group, connection-id)

checkpoint(group, connection-id, checkpoint-id)

report(group, connection-id, member-id, change, source-host)

Connection_monitor() is called at the multicast originator to initiate XGM state information for monitoring the one-to-many data exchange identified by **group** and **connection-id**. **Membership** specifies the required membership in the connection with the exact form of specification dependent on the XTP service interface. At the minimum, specification in terms of subgroup vectors should be supported. The state information at a GME concerning a connection is released as part of the XTP disconnect state machine, which calls **release_monitor()**.

The semantics of the **checkpoint()** primitive are that the membership associated with the XGM internal data structure identified by **group** and **connection-id** be verified via some form of group poll or equivalent mechanism. **Checkpoint-id** differentiates between **checkpoint()** calls, which may take place concurrently with data transfer. The group poll that results from a **checkpoint()** call can take place in a number of ways. One scenario would be to have the Transport Layer user indicate checkpoints via placement of a marker in the data stream. The poll request could travel in an XTP PDU using XTP's *tagged data* channel ([48]). At the receivers, the tagged data will be delivered to the local GME, which will send an out-of-band reply to the GME at the multicast originator. In this case, the **checkpoint-id** could be the XTP sequence number associated with the marker.

When a multicast member drops into or leaves a multicast conversation, the local GME issues a **report()** to the GME at the multicast sender. This reporting facility maintains the integrity of the membership tracking within a connection monitor in the face of shifting membership. A method for joining an in-progress multicast conversation is defined in XTP ([48]). The GME plays a watchdog role in the detection of local multicast contexts that close abnormally. **Change** indicates whether a member has been added to or deleted from the group. **Source-host** should always be known at the remote GME since this information is present at a multicast group member in an active connection.

Chapter 5

Reliable Multicast Service using XTP

5.1. Introduction

In this chapter the ideas behind group management introduced in the previous chapter are explored in the context of the software implementation of the Xpress Transfer Protocol done by the University of Virginia Computer Networks Laboratory. First we present a multicast facility (the Multidriver), constructed over XTP and independent of the XTP multicast algorithm, using the out-of-band *tagged data* channel provided by XTP ([20]). The Multidriver was designed by John Fenton and the author, and Fenton implemented it as part of the UVA XTP driver set ([20]). The Multidriver gathers explicit acknowledgments from all members of the multicast set and thus achieves reliable transfer in a much stronger sense than the XTP multicast algorithm. This thesis extends the Multidriver scheme to support group management, and the functionality and performance of this multicast facility, the List-Based Multidriver, are probed.

5.2. UVA XTP

The University of Virginia Computer Networks Laboratory has implemented the Xpress Transfer Protocol in software. The UVA XTP architecture has a layered structure. The bottom layer represents the 802.2 Logical Link Control (LLC) ([30]) interface to the network. Above the LLC sits the XTP Engine, which performs protocol processing on XTP *contexts*, the structures that hold connection state information at an endpoint. (In UVA XTP each context must either be a transmit or a receive context.) At the highest layer reside *XTP drivers*. Drivers are special purpose modules that use the low-level interface to the Engine to implement an XTP service interface. Engine and driver communicate through shared memory in the context structures, a small set of C subroutines, and upcalls. XTP drivers handle decisions about retransmission, flow control, synchronization, and buffering in order that the Engine performs only the protocol processing common to all XTP users.

The separation of policy (driver) and mechanism (Engine) enables great flexibility in designing the user interface to UVA XTP. Drivers can be tailored to the communication needs of a particular application or class of applications. To facilitate driver development, a set of primitives has been implemented that are functionally modeled on UNIX system calls so as to provide a well-known user interface. From them XTP drivers have been written for several communication services, including file transfer, memory-to-memory transfer, and stream I/O. Driver primitives interoperate so that an application links with a driver library and includes only the code necessary for that application. The code fragment in Figure 5.1 illustrates the use of driver routines. It shows an application that reads characters from the network and displays them until the connection is closed.

5.3. Multidriver

Our multicast facility is implemented as an XTP driver (the Multidriver) that provides the user with four primitives. For unreliable service, the user calls an

```
main( )
{
  XTP_startup( );
  if (xhandle = X_open("pipein","r",device)) < SUCCESS)
    { fprintf(stderr,"Unable to receive from network");
      XTP_finish(-1); }
  while ((c=X_getc(xhandle)) > EOF)
    putchar(c);
  X_close(xhandle);
  XTP_finish(1);
}
```

Figure 5.1 — XTP Driver Primitives

initializing routine to set up state for a multicast transmit context. Data transfer through the context is then available using any of the driver routines. For reliable multicasting, the user calls an initializing routine that carries out a series of actions: (1) it sets up the transmit context; (2) it creates a user-specified number of receive contexts (*response contexts*); (3) it issues a connection set-up packet from the transmit context in order to establish the multicast (forward) connection; and (4) it monitors the contexts set up in (2) to ensure the establishment of some user-controlled number of connections between multicast group members and the response contexts. After these connections are made, the user can carry out a reliable multicast transaction with the set of receivers that have established response channels.

XTP provides a packet of type FIRST, which can carry user data as well as addressing information, to set up a connection. After a FIRST packet establishes the connection, the data source issues DATA packets. In a reliable unicast, the sending context determines when the receiving context will issue CNTL packets, which contain control information, by setting certain request bits in out-going DATA (or FIRST) packets. Within the byte-sequenced data stream of an XTP connection, out-of-band, or *tagged*, data can appear as the first 8 (BTAG header bit set) or the last 8 (ETAG header bit set) bytes of a DATA or FIRST packet. At the remote end, XTP passes up tagged data uninterpreted to the user ([47]). The Multidriver suppresses XTP's error control, e.g., the multicast transmitting context never sets status request header bits in a DATA packet. For reliable multicasting, the Multidriver manages its own control scheme by sending control information as tagged data, which can be multiplexed with user data, in both the forward and reverse channels.

5.3.1. Multidriver Design

The Multidriver design focuses on extending the unicast virtual circuit paradigm to a one-to-many connection. Implementation of this model requires solving synchronization and coordination problems not encountered in unicast protocols. Control information for the multicast connection must be efficiently and effectively collected at the multicast source and there coalesced into directives for the multicast transmit context.

Reliable one-to-many delivery implies the existence of some method for tracking the progress of a set of receivers. Otherwise, the multicast sender cannot provide reliable delivery since it cannot detect lagging or failed receivers. Rather than constructing its own method for handling the control flow from multiple data sinks, the Multidriver uses a well-defined mechanism already available within the Xpress Transfer Protocol — XTP connections. Unicast connections to response contexts create channels for driver-level control information as well as client data.

Laminating together XTP connections is attractive for a number of reasons. First, XTP supports rapid connection set-up and tear-down. An XTP FIRST packet can establish a connection and carry user data (as well as deliver tagged data). Connection tear-down involves a 2- or 3-packet handshake that is initiated by the final DATA packet in the transfer. Second, the mapping of individual receivers to their response channels takes place dynamically as in-coming FIRST packets establish connections with response contexts; no prior coordination or management is needed. Finally, since control communication can be handled using tagged data, the side channels enable bi-directional user data flows, i.e., multicast transactions. Reply handling in client/server interactions has been recognized as an important component of multicast communication in many classes of applications ([40]).

For many-to-one data flows, particularly within a LAN, the phenomenon of *network implosion* must be addressed. Implosion refers to the tendency of multicast receivers to synchronize the sending of their control packets in any transmitter-driven scheme. Synchronized transmission can result in bursts of traffic on the network and the inability of the multicast source's network interface to capture frames arriving back-to-back. Since the Multidriver supports the gathering of user-level responses from multicast group members, the problem of coordinating the reverse channels grows with the product of the amount of data in the reverse channel from each receiver and multicast set size. The Multidriver implements mechanisms that allow the multicast source to control network implosion. The administrator of implosion control policy, whether a human user or a management protocol, can use these mechanisms to determine the appropriate implosion control strategy. The synchronization issues involved with network implosion are highly dependent on system parameters. Hence the appropriate strategy is for the multicast communication facility to provide the user with parameters that can be tuned to the target environment.

The Multidriver design implements error, flow, and rate control for the multicast connection. The multicast source solicits control parameters from the set of receivers in the exchange. At the multicast source, after each receiver has responded, the Multidriver takes the minimum of the reported control parameter values and submits this information to the multicast transmit context in the form of an XTP CNTL packet. Since the multicast transmit context is unaware that CNTL packets are being manufactured from above (by the Multidriver) instead of arriving from below (from the network), protocol processing inside the XTP Engine takes place exactly as with reliable unicasting. In this way, control information from multiple communication endpoints is coalesced into directives for controlling the multicast transfer without the addition of extra checks within the transmit Engine.

Error control uses a go-back-N retransmission strategy as selective retransmission for multicast in a LAN environment (e.g., low latencies and relatively high bandwidths) seems unjustifiably complex. The Multidriver releases data in the transmit buffer as soon as arriving control information indicates that all receivers in the exchange have that data. Flow and rate control policies conform to the smallest values reported from the receiver group since faster transfer will only result in costly errors due to dropped packets.

5.3.2. Multidriver Service Primitives

X_Mopen(name, mode, device)

Depending on the value of `mode`, this routine either opens a multicast context for unreliable multicast transmission or opens a context for multicast reception. In the latter case, `X_Mopen()` opens a receive context that listens on the group address associated with `name`.

For unreliable multicast transmission, a transmit context is initialized such that the header bits for multicast (MULTI) and datagram transmission (NOERR) will be set in all FIRST and DATA packets issued from the context. Header bits requesting CNTL packets are guaranteed not to be set in any out-going packet. Otherwise, `X_Mopen()` performs the same state initialization as its unicast counterpart, `X_Open()`. `X_Mopen()` returns a handle of type XFILE that the user must use in driver calls to identify the opened XTP context.

The parameter `name` has a string value identifying a multicast group. The string is mapped internally to the addresses that identify the set of listeners that make up the multicast group. These addresses include the medium dependent hardware address, typically a group address, and the transfer layer address. The format of the transfer layer address depends on the environment since XTP supports multiple addressing modes. The parameter `device` is present since a single implementation of XTP can multiplex between multiple network interfaces.

X_MRopen(group, response, device, min, max, &xfiles[])

`X_MRopen()` performs a series of actions. It sets up a transmit context to send to the multicast address to which `group` maps, and it initializes `max` receive contexts to listen on the address to which `response` maps. A FIRST packet is issued from the transmit context with tagged data containing the value of the parameter `response`, and a timer initialized. Upon expiration of the timer, `X_MRopen()` returns to the user with an error indication if fewer than `min` multicast set members have established connections with local response contexts. Otherwise, `X_MRopen` returns a XFILE handle to the multicast transmit context and places a XFILE handle in the array `xfiles[]` for each active response context, up to a limit of `max`.

X_MRclose(xfile, &xfiles[])

`X_MRclose()` closes a reliable group transmit context, `xfile`, and its associated response contexts, `xfiles[]`.

X_MRreply(xfile)

The multicast receiver opens a receive context (using `X_Mopen()`) that listens on the group address. Upon the arrival of a FIRST packet, the Multidriver checks the ETAG header bit. If ETAG is set, the Multidriver opens a transmit context (the *return context*) and sends a FIRST packet to the address to which the string in the ETAG field maps (see Figure 5.2).

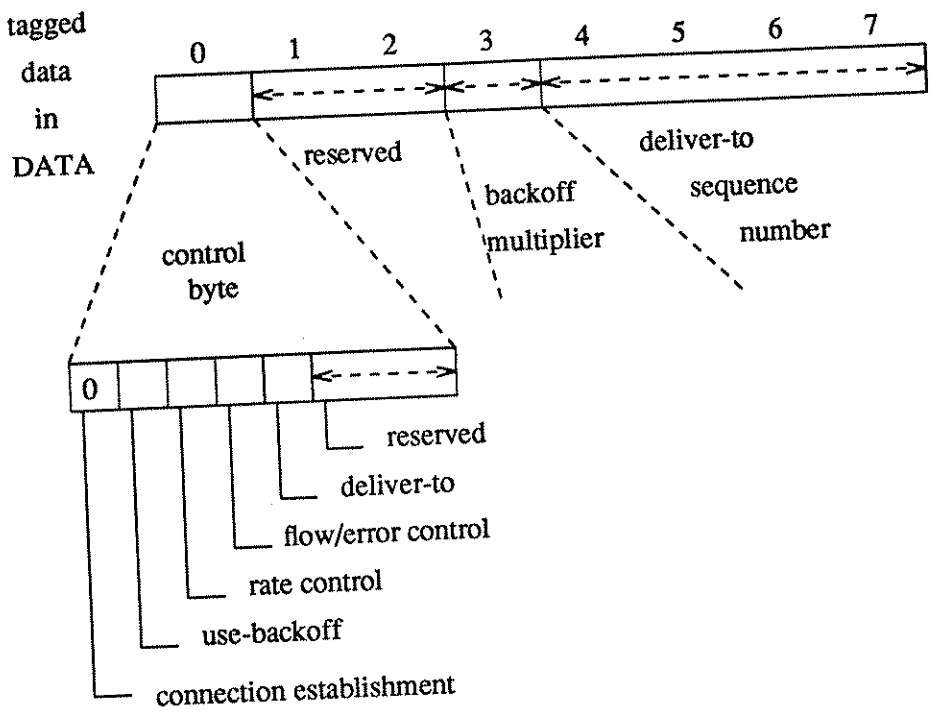
`X_MRreply()` allows the local client process to send data to the multicast source using the return context. The parameter `xfile` provides the handle of the multicast receive context, not the return context, since the return context is managed completely internally by the Multidriver. `X_MRreply()` returns a special handle of type KEY for the return context. The KEY handle can be used in place of an XFILE handle in driver routines, which check for this special case. With the exception of this indirection and the loss of the tagged data feature, the return channel functions as an ordinary XTP reliable unicast connection.

The multicast receive context and its return context are always closed together and can be closed in two ways. Either the remote end transmits a close indication to the receiving context or the return context, or the local user closes the receiving context. The local user cannot close the return context directly.

5.3.3. Control Scheme

All tagged data in a reliable multicast transaction exchange represents driver-level control information, which is embedded in both the forward and reverse data streams. At a multicast group member, if the local application process does not generate reverse direction data, then the return context will be issuing XTP DATA packets containing only tagged data. For tagged data in the source-to-group (forward) direction, the first byte of the the tagged data field serves as a control byte (see Figure 5.2). In the reverse direction, no control byte is needed. ETAG fields carry responses to the flow/error control flag (see below), and BTAG fields carry rate control parameters.

(1) connection establishment flag — indicates the presence in the tagged data field of two items: first, a string that maps to the address on which to open the return context



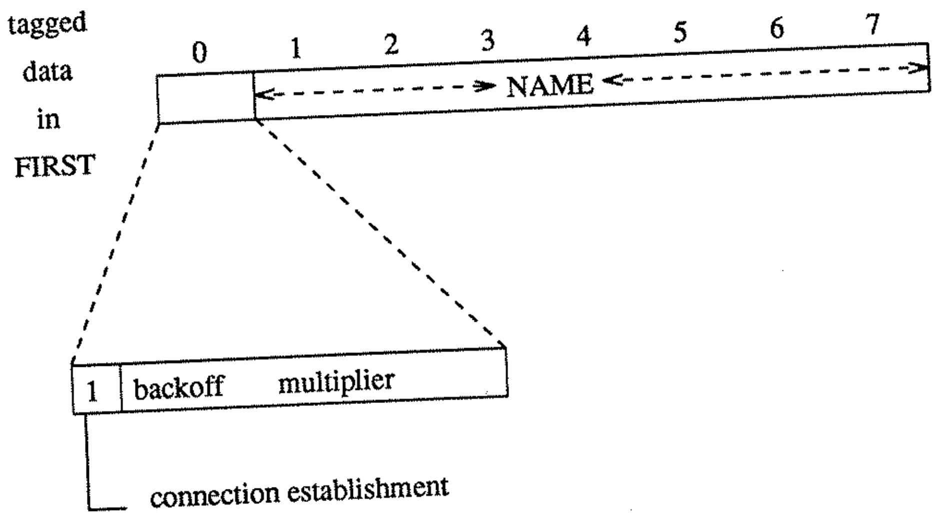


Figure 5.2 — Multidriver Control Scheme

and second, an integer denoting multicast set size. Before transmitting the FIRST packet from the return context, the Multidriver receiver waits a random amount of time between 0 and BACKOFF_TIME, which is determined by the multicast set size.

(2) flow/error control flag — requests the remote end to report one plus the sequence number of the last byte received in order at the receiving context and one plus the sequence number of the last byte that the receiving context will accept.

(3) rate control flag — requests the remote end to report its XTP rate control parameters, BURST and RATE.

(4) deliver-to flag — notifies the remote end not to deliver data to the destination process beyond the enclosed sequence number. The deliver-to flag offers the multicast source a mechanism for synchronizing message commitment.

(5) use-backoff flag — delivers an integer used for BACKOFF_TIME computation. The flag indicates that a multicast receiver must use the accompanying integer to compute the value of its BACKOFF_TIME variable and begin using a random backoff between 0 and BACKOFF_TIME for each packet transmitted from the return context.

5.4. List-Based Multidriver

The List-Based Multidriver (LBM) is an extension of the Multidriver scheme that gives the user more control over the multicast transfer. Under LBM, a multicast transmit context has an explicit list of receivers, a *transfer list*, associated with it. The application may fill in the transfer list from a private address table or, as with the Multidriver, the list can be built from the network during a group connection set-up phase. Each multicast context has a user-settable subgroup vector associated with it. For receive contexts, the subgroup vector serves as a filter on the set of in-coming connections to which it will belong. For a transmit context, the subgroup vector determines the set of group members that will participate in a reliable one-to-many data transfer.

The contents of the control tags are the same for LBM as for the Multidriver except that the subgroup vector of the context issuing the tag is included in the tag. Tags in the many-to-one direction are now sent as reliable datagrams, breaking the one-to-one association between members of the receiver set and response contexts. At the multicast originator, a set of response contexts are available to accept and acknowledge in-coming FIRST packets that will contain control tags. These contexts acknowledge the receipt of the packet. LBM logic reads the control tag and updates the transfer list. It then closes the response context and opens a replacement context.

The LBM strategy of having a transfer list hold the state information of the multicast set trades off a reverse data channel per receiver for greater flexibility in group management and XTP resource allocation. On the latter point, the number of contexts available in UVA XTP is currently 16 and will be 32 when UVA XTP is updated to reflect XTP Revision 3.5. These values are significant in the sense that they allow certain atomic bit mask operations that lend tremendous efficiency gains to the implementation as a whole. While the target environments for UVA XTP have small enough node populations that 32 contexts seems adequate, multicast groups could grow to a size where having a response context dedicated to each receiver would not be feasible.

LBM has been implemented in a modular fashion so that all the Multidriver primitives and semantics can coexist with it. During data transfer by any of the XTP driver primitives, if a multicast transmit context has no transfer list, the Multidriver's scheme for processing control tag information is followed. Otherwise, LBM processing takes place.

The primitives for LBM include:

G_Subgroup(xfile, &subgroup, mode)

This primitive allows the subgroup vector of a context to be set or read, depending on the value of `mode`. In our implementation we allow 8 subgroups. `X_Mopen()` has been modified such that it selects a random subgroup with which to associate the multicast receive context being opened. For transmit contexts, the subgroup vector is set using the `G_MRopen()` primitive. `G_Subgroup()` allows the user to change the subgroup vector in order to shift a receive context into another subgroup or to change the reliability criterion for data transfers from a transmit context.

G_Members(xfile, &memberlist, mode)

This primitive allows the user to read or write the membership list of a multicast transmit context, `xfile`. The `memberlist` is a linked list of entries, each of which has two fields, a 48-bit IEEE compatible MAC layer node address and a byte representing the subgroup vector. The `memberlist` is copied into the transfer list for `xfile` if the `mode` is WRITE, and it is returned with the contents of the current transfer list for `xfile` if the mode is READ.

G_MRclose(xfile)

`G_MRclose()` closes a reliable group transmit context, `xfile`, and its associated response contexts.

G_MRopen(group, response, subgroups, device, &min, max)

The parameters `group`, `response`, and `device` have the same functions for `G_MRopen()` as for `X_MRopen()`. `G_MRopen()` performs group set-up by sending to the address `group` a FIRST packet with a tag that includes the subgroup vector `subgroups` as well as the other control information. A receiver only responds to the multicast FIRST packet if its subgroup is among those in `subgroups`. The multicast sender opens `max` response contexts on the `response` address. At least `min` receivers must respond and the actual number that respond is returned in `min`. `Max` response contexts will be retained for the duration of the group connection.

As receivers respond, they are entered into the transmit context's transfer list. Each responding control tag contains the subgroup of the responding member, and the transfer list clusters members by subgroup. As with `X_MRopen()`, this process continues until a timer expires.

`G_MRopen()` imposes an initial selection process on group formation using the `subgroups` parameter. LBM also offers the flexibility of pruning the group after the group formation phase. After examination of the transfer list by a call to `G_Members()`, the user may reset the subgroup vector of the transmit context using `G_Subgroup()`. To suppress extraneous network traffic, LBM always transmits the subgroup vector of the transmit context in the initial packet containing user data. Each receiver checks this packet to see if its subgroup has been pruned. If so, the receiver leaves the conversation immediately. Excluded members close their contexts with an indication to the local user. Since receivers are not associated with specific response contexts, the loss of receivers that were entered into the transfer list during connection set-up does not alter the number of response contexts available at the multicast originator for many-to-one control flow.

5.5. Performance

In this section we present performance numbers for the Multidriver and LBM over a heterogeneous single-segment network.

5.5.1. Target Environment

UVA XTP is designed to serve as the transfer layer component for a real-time communications subsystem such as that specified in the SAFENET standards for military and commercial ships ([27]). In the UVA implementation, XTP runs on top of a real-time, link layer messaging service ([52]). The performance measurements below were done on a single-segment Proteon ProNET-4 802.5 token ring operating at 4 Mbit/s. Network nodes include ALR 25 MHz Intel 386 FlexCaches (Flexs), Zenith 16 MHz Intel 386 machines (Zeniths), and Core 25 MHz Intel 386 machines (Cores). All nodes have AT buses.

5.5.2. Multicast Latency

To measure multicast latency, we perform a number (typically 100) of consecutive reliable 1-byte data transfers and measure the average message latency. Multicast latency is defined as the time between submission of the message to the XTP driver level and confirmation that all members in the multicast exchange received the message. The timing routines used are believed to be accurate to within a few milliseconds; they have been verified to be accurate within 50 ms by comparison with the packet timestamping of WireTap ([43]), a real-time network monitor developed by the University of Virginia Computer Networks Laboratory. Though crude, the upper bound on timing error provided by WireTap ensures a maximum timing error of less than 10% for all individual measurements and less than 3% in most cases. Timing ambiguities are further smoothed by taking the average of at least three separate runs of the test program. The numbers given in the tables below are such averages.

Table 5.2 presents the unicast user latency for one-byte messages from a Flex to each of the other types of machines. Table 5.1 presents the LBM multicast latency from the Flex to receiver sets varying from 1 to 5 nodes. The figures show that the LBM scheme imposes a much higher latency cost than unicast, but latency costs increase slowly as the size of the receiving set grows. For a Flex-to-Zenith transfer, multicast latency is over three times the unicast latency. As the number of nodes increase, however, the multicast latency grows slowly, more slowly than the increase for a series of unicast transfers to the same receiver set. By the time a 5-node receiver set is reached, the multicast latency, 36.3 ms, is within 20% of the sum of the unicast transfers, 30.5 ms.

Multicast Roundtrip Latency (LBM) Transmitting Node: 25 MHz 80386 Flex 1-byte messages		
Number of Receiving Nodes	*Receiving Nodes*	*Average Roundtrip Latency*
1	1 Zenith	19.8 ms
2	2 Zeniths	23.1 ms
3	2 Zeniths, 1 Core	27.1 ms
4	2 Zeniths, 2 Cores	31.8 ms
5	2 Zeniths, 2 Cores, 1 Flex	36.3 ms

Table 5.1 — LMB Roundtrip Latency

By extrapolation, the experimental data suggests that the LBM multicast scheme offers lower latencies only in the case where the number of nodes in the multicast group is relatively large (e.g., generally greater than 8-10) and/or the unicast latency is high with some members of the multicast group.

The relatively high minimum latency of the LBM scheme reflects its reliance on multiple crossings of the boundary between Engine and Driver. Unicast latencies measure Engine-to-Engine interaction since, upon reception of the FIRST packet containing the byte of user data, the receiving Engine sends back an XTP CNTL packet. The transmitting Engine receives the CNTL packet and signals the user, and the clock is stopped. In contrast, under the Multidriver, the arriving DATA packet at the remote end contains tagged data, which must be delivered to the driver level. The receiving Multidriver constructs a DATA packet containing the driver-level acknowledgement and transmits it from the return context. This reverse-direction DATA packet is acknowledged with an XTP CNTL packet at the original transmitter's response context. Both the response context and the multicast transmission context are updated by the

Unicast Roundtrip Latency	
Transmitting Node: 25 MHz 80386 Flex	
1-byte messages	
Receiving Node	*Average Roundtrip Latency*
Flex	5.7 ms
Core	5.6 ms
Zenith	6.8 ms

Table 5.2 — Unicast Roundtrip Latency

Multidriver before the user is notified and the clock stopped.

Table 5.3 shows the Multidriver latencies for the same transmitter and receiver sets as in Table 5.1. The LBM scheme is more expensive than the Multidriver scheme in that the reliance on reliable datagrams from the receivers forces the opening and closing of contexts as tags are processed. Table 5.3 confirms that, at least with respect to message latency, the overhead for the greater flexibility of LBM is negligible.

5.5.3. Multicast Throughput

Table 5.4 shows the achievable throughput for LBM in delivering a large file (500 Kbytes) reliably to a multicast group. The message buffers are 16 Kbytes for all contexts, except those involved in the reliable datagram exchange for control messages. The latter need only very small, typically 128-byte, buffers. Using the same driver routines, unicast transfers from the same transmitting node (a Zenith) to the other node types achieved the throughputs shown in Table 5.5.

Multicast Roundtrip Latency (Multidriver)
Transmitting Node: 25 MHz 80386 Flex
1-byte messages

Number of Receiving Nodes	*Receiving Nodes*	*Average Roundtrip Latency*
1	1 Zenith	17.5 ms
2	2 Zeniths	22.5 ms
3	2 Zeniths, 1 Core	26.7 ms
4	2 Zeniths, 2 Cores	31.8 ms
5	2 Zeniths, 2 Cores, 1 Flex	37.0 ms

Table 5.3 — Multidriver Roundtrip Latency

Multicast File Transfer (LBM)		
Transmitting Node: 16 MHz Intel 386 Zenith		
Receiving Nodes	*Receiver Set*	*Average Throughput*
1	1 Flex	273.8 Kbits/s
2	1 Flex, 1 Zenith	235.3 Kbits/s
3	2 Flex, 1 Zenith	225.4 Kbits/s
4	2 Flexs, 1 Zenith, 1 Core	206.9 Kbits/s
5	2 Flexs, 1 Zenith 2 Cores	191.1 Kbits/s

Table 5.4 — LBM Throughput

Unicast Throughput	
Transmitting Node: 16 MHz 80386 Zenith	
Receiving Node	*Average Throughput*
Flex	606.1 Kbits/s
Core	640.0 Kbits/s
Zenith	347.8 Kbits/s

Table 5.5 — Unicast Throughput

Multicasting to a group containing only one receiver causes a 55% drop in throughput when compared to unicasting. In the two-receiver case, however, reliable multicasting yields essentially the same throughput as sequential unicasting. To see this, consider that the average of the two nodes' unicast throughputs is 476.9 Kbits/s; hence sequential unicast delivery to the two nodes will yield an overall transfer rate of one half the average, or 238.5 Kbits/s. As shown in Table 5.4, the multicast throughput is 235.3 Kbits/s. With each node after the second, the advantage of multicasting grows rapidly. At five nodes, the sequential unicast rate is 113.6 Kbits/s, only 59% of the multicast rate of 191.1 Kbits/s. These figures indicate that for bulk data movement the LBM scheme can achieve substantially better performance than unicast transfers and that these efficiencies are realized as soon as more than two hosts have joined the multicast group.

As shown in Table 5.5, the use of the slower machine, a Zenith, as a receiver causes a substantial drop in potential unicast throughput. (This unfortunate situation is being rectified by a more robust buffer strategy as the UVA XTP implementation is being updated to XTP Revision 3.5.) Multicast throughput suffers a similar decrease, as indicated by Table 5.6. This table shows the throughputs attainable when the FlexCache is used as a transmitter and the Zeniths as receivers. Like the Zenith transmitter, the Flex can unicast to a Zenith at roughly 350 Kbits/s. The multicast throughput to a single Zenith reflects a drop of about 45% from unicasting. With two Zeniths in the multicast set, multicast and sequential unicast become essentially equivalent with rates of 175.7 Kbits/s and 173.9 Kbits/s, respectively. Hence our experimental results suggest that the relative performance gains of the LBM are stable with respect to the relative processing power of the multicast transmitter and its receivers.

Table 5.7 shows that the Multidriver outperforms LBM by as much as 15%. While the Multidriver has greater throughput for every receiver set shown, the discrepancy

Multicast File Transfer (LBM)		
Transmitting Node: 25 MHz Intel 386 FlexCache		
Receiving Nodes	*Receiver Set*	*Average Throughput*
1	1 Zenith	189.5 Kbits/s
2	2 Zeniths	175.7 Kbits/s

Table 5.6 — LBM Throughput using a FlexCache

Multicast File Transfer (Multidriver)		
Transmitting Node: 16 MHz Intel 386 Zenith		
Receiving Nodes	*Receiver Set*	*Average Throughput*
1	1 Flex	320.0 Kbits/s
2	1 Flex, 1 Zenith	241.7 Kbits/s
3	2 Flex, 1 Zenith	233.4 Kbits/s
4	2 Flexs, 1 Zenith, 1 Core	230.0 Kbits/s
5	2 Flexs, 1 Zenith 2 Cores	221.4 Kbits/s

Table 5.7 — Multidriver Throughput

between the two schemes ranges from 2.6% to 14.4%. Curiously, the maximum difference as measured by percentage appears in both the single node and the five node case. This decrease in throughput can be taken to be the price paid by the LBM for manipulating the transfer list and opening and closing contexts for the transmission and reception of each control tag.

Chapter 6

Conclusions

6.1. Summary

As classically defined, the Transport Layer turns the underlying unreliable service of the Network and Data Link Layers into a reliable service. A Transport Layer reliable multicast facility taps the power of the underlying broadcast and multicast hardware found in many modern LANs for efficient multidestination delivery of messages. With multicasting, the term *reliable* denotes error-free, in-order delivery of the Transport Layer Service Data Unit to the 'correct' set of destinations. Since multicast addresses commonly represent a logical set whose membership is determined by run-time bindings, the notion of a reliable multicast necessarily implies *multicast group management* mechanisms.

The range of distributed applications that would benefit from a reliable multicast facility is sufficiently diverse to insure that no single multicast facility will be right for all communication environments. For tightly coupled distributed processing groups in which each member of the group may send messages to the other members, powerful atomic multicast protocols have been proposed to ensure the atomic delivery and identical ordering of messages to the group. For these protocols, reliability mechanisms at the Transport Layer are generally not useful. On the other hand, many distributed applications can be expected to exhibit a client-server communication pattern in which there is one, or a few, data sources and a set of data sinks. These applications will need primitives at the Transport Layer for determining and controlling the set of destinations for a particular data exchange or series of exchanges.

This thesis proposes a service interface for a group management facility to complement the Xpress Transfer Protocol's multicast algorithm. It then details the incorporation of group management functionality into the Multidriver, a reliable multicast service built into UVA XTP that is independent of the multicast algorithm in the XTP protocol specification. Performance measurements of the Multidriver and the List-Based Multidriver allow the quantification, within the testbed environment for UVA XTP, of performance characteristics for these two facilities.

These measurements verify that multicasting offers significant performance advantages over sequential unicasting. In our experiments the Multidriver schemes achieve greater throughputs than sequential unicasting as soon as there are more than two nodes in the receiving set. With five nodes receiving, the LBM improves the throughput available from sequential unicast delivery by 68%. Measurements of the roundtrip latency for short messages indicate that, for small multicast groups, the Multidriver schemes can not match the delivery rates of sequential unicasts. As nodes are added to the multicast group, however, the multicast latency performance converges rapidly towards that of sequential unicasts. While with one receiving node the multicast latencies measured were close to 300% of the unicast latencies, the performance gap has closed to only a 20% difference by the time the multicast set has grown to five nodes. By extrapolation, when the receiving set enters the range of 6 to 10 nodes, multicast latencies can be expected to best those of sequential unicasts.

The Multidriver and its extension for list-based transfers performed comparably with respect to both latency and throughput measurements. Latency measurements were quite similar with the exception of the one-node multicast case, for which the Multidriver was 12% faster than the LBM. With respect to throughput capability, the Multidriver consistently outperformed the LBM, though again the performance differences were slight. The Multidriver achieved from 2.6% to 14.4% greater throughput for the receiver sets tested. This difference can be attributed to the overhead imposed on control tag processing by the LBM's use of reliable datagrams, instead of connections, for control flow in the many-to-one direction. As noted above, the LBM is nonetheless more efficient than sequential unicasting in the multidestination delivery of large messages when the receiving set consists of more than two nodes.

6.2. Future Work

Research in the area of reliable multicasting remains somewhat immature. Experience with large sets is particularly rare. With the porting of UVA XTP to Ethernet, which is in progress, and the support of UVA Academic Computing, we have the unique opportunity to study the performance of the Multidriver and the LBM over heavily populated networks. Especially interesting issues to probe include: (1) to see how gracefully and to what point the mechanisms in the Multidriver scheme will scale, (2) to investigate algorithms to deal with the problem of network implosion, (3) to explore the effect on multicasting of the improved buffer strategies within UVA XTP, and (4) to measure directly the performance differences between the XTP multicast algorithm, which will be implemented in UVA XTP, and the Multidriver schemes. This work should lead to additional insights into reliable multicasting, insights that are very difficult to glean from small node populations.

REFERENCES

1. L. Aguilar, ''Datagram Routing for Internet Multicasting'', *Computer Communications Review (USA) 14*, 2 (1984), 58-63 .
2. M. Ahamad, M. H. Ammar, J. M. Bernabeu-Arban and M. Khalidi, ''Using Multicast Communication to Locate Resources in LAN-Based Distributed System'', *Proceedings of the 13th Conference on Local Computer Networks*, Minneapolis, Minnesota, 1988.
3. *FDDI Token Ring Media Access Control Standard*, American National Standards Institute, Feb. 1986. Draft proposed Standard X3T9.5/83-16, Rev. 10.
4. S. Andersen, Multicast Connection Oriented Services, PEI Document 90-81, June 1990.
5. F. Backes, ''Transparent Bridges for Interconnection of IEEE 802 LANs'', *IEEE Network 2*, 1 (January 1988).
6. K. Birman and T. Joseph, ''Reliable Communication in the Presence of Failures'', *ACM Transactions on Computer Systems 5*, 1 (February 1987), 47-76.
7. J. Chang and N. F. Maxemchuk, ''Reliable Broadcast Protocols'', *ACM Transactions on Computer Science 2*, 3 (Aug. 1984), 251-273.
8. D. R. Cheriton and S. E. Deering, ''Host Groups: A Multicast Extension for Datagram Internetworks'', *Proc. of the Ninth Data Communications Symposium*, Whistler Mountain, BC, Canada, Sep. 1985, 172-179.
9. D. R. Cheriton and W. Zwaenepoel, ''Distributed Process Groups in the V Kernel'', *ACM Transactions on Computer Systems 3*, 2 (May 1985), 77-107.
10. D. Cheriton, *VMTP: Versatile Message Transaction Protocol -- Protocol Specification*, Stanford University, February 1988. Version 0.7.
11. D. Cheriton and C. L. Williamson, ''VMTP as the Transport Layer for High-Performance Distributed Systems'', *IEEE Communications Magazine*, June 1989, 37-44.
12. G. Chesson, ''The Protocol Engine Project'', *Unix Review*, September 1987.
13. M. Cohn, ''Functional Addressing: Another Way of Looking at Multicast'', *Transfer 2*, 6 (November/December 1989), 13-15.
14. D. Comer, *Internetworking with TCP/IP*, Prentice-Hall, Englewood Cliffs, New Jersey, 1988.
15. E. C. Cooper, ''Circus: A Replicated Procedure Call Facility'', *Fourth Symposium on Reliability in Distributed Software and Database Systems*, 1984.

16. J. Crowcroft and K. Paliwoda, "A Multicast Transport Protocol ", *CCR 18*, 4 (Aug. 1988), 247-256.

17. Y. K. Dalal and R. M. Metcalfe, "Reverse Path Forwarding of Broadcast Packets", *Comm. of the ACM 21*, 12 (Dec. 1978), 1040-1048.

18. S. E. Deering and D. Cheriton, "Multicast Routing in Datagram Internetworks and Extended LANs", *Computer Communications Review 18*, 4 (August 1988).

19. S. E. Deering, *Host Extensions for IP Multicasting: RFC 1112*, August 1989.

20. B. J. Dempsey, J. C. Fenton and A. C. Weaver, "The Multidriver: A Reliable Multicast Service Using the Xpress Transfer Protocol", *15th Conference on Local Computer Networks*, Minneapolis, Minn., October 1990, 351-358.

21. B. J. Dempsey and A. C. Weaver, Multicast Strategies for XTP, PEI Document 90-5, January 1990.

22. *The Ethernet: A Local Area Network — Data Link Layer and Physical Layer Specifications*, Digital Equipment Corporation, Intel Corporation, Xerox Corporation, November 1982.

23. R. Dixon and D. Pitt, "Addressing, Bridging, and Source Routing ", *IEEE Network 2*, 1 (January 1988).

24. J. Foley, *X3T5 Chair*, personal conversation, September 1990.

25. A. J. Frank, L. D. Wittie and A. J. Bernstein, "Multicast Communication on Network Computers", *IEEE Software*, May 1985, 49-61.

26. H. Garcia-Molina and A. Spauster, "Message Ordering in a Multicast Environment ", *Proceedings of the Ninth International Conference on Distributed Computer Systems*, Newport Beach, California,, June 1989, 354-361 .

27. D. T. Green and D. T. Marlow, "SAFENET — A LAN for Navy Mission Critical Systems", *Proc. of the 14th Conference on Local Computer Networks*, Minneapolis, Minnesota, October 1989.

28. E. Harris, *Vice-Chair X3T5.4*, personal conversation, August 1990.

29. *IEEE Standard 802.1D MAC Bridges*, Institute for Electrical and Electronic Engineers Project 802— Local and Metropolitan Area Network Standards , 1985 .

30. *IEEE Standard 802.2 Logical Link Control*, Institute of Electrical and Electronics Engineers, 1984.

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

32. *IEEE Standard 802.4 Token-Passing Bus Access Method and Physical Layer Specifications*, Institute of Electrical and Electronics Engineers, 1985.

33. *IEEE Standard 802.5 Token Ring Access Method and Physical Layer Specifications*, Institute of Electrical and Electronics Engineers, 1985.

34. "Information Processing Systems - Open Systems Interconnection - Basic Reference Model", *Draft International Standard 7498*, Oct. 1984.

35. ''Common Management Information Service (CMIS) Definition'', *Draft International Standard 9595*, June 1989.

36. ''Common Management Information Protocol (CMIP) Specification'', *Draft International Standard 9596*, June 1989.

37. M. F. Kaashoek, A. S. Tanenbaum, S. F. Hummel and H. E. Bal, ''An Efficient Reliable Broadcast Protocol '', *Operating Systems Review 23*, 4 (October 1989).

38. J. Kramer, J. Magee and A. Lister, ''CONIC: An Integrated Approach to Distributed Computer Control Systems'', *IEE Proceedings Part E 130*, 1 (January 1983), 1-10.

39. J. Lederburg and K. Uncapher, ''Towards a National Collaboratory'', *Report of an Invitational NFS Workshop*, March 1989.

40. L. Liang, S. T. Chanson and G. W. Neufield, ''Process Groups and Group Communications: Classifications and Requirements'', *IEEE Computer 23*, 2 (February 1990), 56-66.

41. D. T. Marlow, ''Requirements for a High Performance Transport Protocol for Use on Naval Platforms'', Revision 1, Naval Surface Warfare Center, July 1989.

42. P. McKinley and J. Liu, " ''Multicast Tree Construction in Bus-Based Computer Networks"'', *Communications of the ACM 33*, 1 (January 1990), 29-42.

43. J. F. McNabb and A. C. Weaver, ''A Real-Time Network Performance Monitor for Token Rings'', *MILCOM 89*, Boston, Mass., October 1989.

44. J. Moulton, ''OSI Networking Paradigm Shift: Next Generation Transport Protocols'', *Transfer 3*, 2 (March/April 1990), 11-18.

45. J. Moulton, *X3S3 member*, personal conversation, September 1990.

46. R. Perlman, A. Harvey and G. Varghese, ''Choosing the Appropriate ISO Layer for LAN Interconnection'', *IEEE Network 2*, 1 (January 1988), 81-85.

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

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

49. B. Rajagopalan and P. McKinley, ''A Token-Based Protocol for Reliable, Ordered Multicast Communication'', *Proceedings of Eighth Symposium on Reliable Distributed Systems* , Seattle, Washington , October 1989.

50. S. Ramakrishnan and B. Jain, ''A Negative Acknowledgement with Periodic Polling Protocol for Multicast over LANs'', *IEEE INFOCOM 1987: The Conference on Computer Communications Proceedings*, San Francisco, California, April 1987.

51. R. Sanders, *The Xpress Transfer Protocol (XTP): A Tutorial* , University of Virginia, 1989 .

52. R. Simoncic, A. C. Weaver, B. G. Cain and M. A. Colvin, ''SHIPNET: A Real-time Local Area Network for Ships'', *Proc. of the 13th Conference on Local*

Computer Networks, Minneapolis, Minnesota, October 1988.

53. W. D. Sincoskie and C. J. Cotton, ''Extended Bridge Algorithms for Large Networks'', *IEEE Network 2*, 1 (January 1988), 16-24.

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

55. *TMS 380 Adapter Chipset User's Guide, Revision D* , Texas Instruments , 1986.

56. D. W. Wall, ''Mechanisms for Broadcast and Selective Broadcast (excerpts from)'', 190, Computer Systems Laboratory, Stanford University, June 1980.

57. W. A. Wulf, *The National Collaboratory — A White Paper* , National Science Foundation , December 1988 .