ISSUES IN PROVIDING A RELIABLE MULTICAST FACILITY

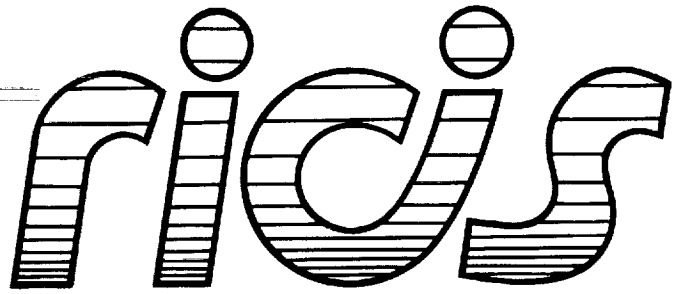
Bert J. Dempsey
W. Timothy Strayer
Alfred C. Weaver

Digital Technology

July 1990

Cooperative Agreement NCC 9-16
Research Activity No. SE.31

NASA Johnson Space Center
Engineering Directorate
Flight Data Systems Division



Research Institute for Computing and Information Systems
University of Houston - Clear Lake

T·E·C·H·N·I·C·A·L R·E·P·O·R·T

The RICIS Concept

The University of Houston-Clear Lake established the Research Institute for Computing and Information systems in 1986 to encourage NASA Johnson Space Center and local industry to actively support research in the computing and information sciences. As part of this endeavor, UH-Clear Lake proposed a partnership with JSC to jointly define and manage an integrated program of research in advanced data processing technology needed for JSC's main missions, including administrative, engineering and science responsibilities. JSC agreed and entered into a three-year cooperative agreement with UH-Clear Lake beginning in May, 1986, to jointly plan and execute such research through RICIS. Additionally, under Cooperative Agreement NCC 9-16, computing and educational facilities are shared by the two institutions to conduct the research.

The mission of RICIS is to conduct, coordinate and disseminate research on computing and information systems among researchers, sponsors and users from UH-Clear Lake, NASA/JSC, and other research organizations. Within UH-Clear Lake, the mission is being implemented through interdisciplinary involvement of faculty and students from each of the four schools: Business, Education, Human Sciences and Humanities, and Natural and Applied Sciences.

Other research organizations are involved via the "gateway" concept. UH-Clear Lake establishes relationships with other universities and research organizations, having common research interests, to provide additional sources of expertise to conduct needed research.

A major role of RICIS is to find the best match of sponsors, researchers and research objectives to advance knowledge in the computing and information sciences. Working jointly with NASA/JSC, RICIS advises on research needs, recommends principals for conducting the research, provides technical and administrative support to coordinate the research, and integrates technical results into the cooperative goals of UH-Clear Lake and NASA/JSC.

ISSUES IN PROVIDING A RELIABLE MULTICAST FACILITY

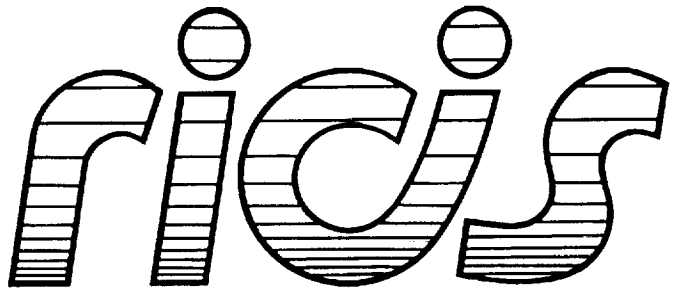
Bert J. Dempsey
W. Timothy Strayer
Alfred C. Weaver

Digital Technology

July 1990

Cooperative Agreement NCC 9-16
Research Activity No. SE.31

NASA Johnson Space Center
Engineering Directorate
Flight Data Systems Division



Research Institute for Computing and Information Systems
University of Houston - Clear Lake

T·E·C·H·N·I·C·A·L R·E·P·O·R·T

Preface

This research was conducted under auspices of the Research Institute for Computing and Information Systems by Bert J. Dempsey, W. Timothy Strayer, Alfred C. Weaver and Digital Technology. Dr. George Collins, Associate Professor of Computer Systems Design, served as RICIS technical representative for this activity.

Funding has been provided by the Engineering Directorate, NASA/JSC through Cooperative Agreement NCC 9-16 between NASA Johnson Space Center and the University of Houston-Clear Lake. The NASA technical monitor for this activity was Frank W. Miller, of the Systems Development Branch, Flight Data Systems Division, Engineering Directorate, NASA/JSC.

Table Of Contents

1. Introduction

Multicasting, or point-to-multipoint communication, enables a message to be delivered from its entry point into the communications system to some well-defined set of destinations. Multicast facilities in networks, mechanisms that enable multi-destination delivery using a single address, provide efficiency and robustness for one-to-many communication. If no multicast facility exists, the alternatives available to effect multi-destination delivery are sequential unicasts to each destination or broadcasting to all hosts. Multicasting is an improvement over a series of unicast transfers because the source generates only a single packet, rather than one per receiver. In contrast to broadcasting, multicasting eliminates the overhead of hosts that are not interested in the distribution having to receive it. These properties of multicast save processing cycles at the source node, bandwidth, and remote host resources. Since the savings accumulate on a *per packet* basis, large messages or messages to be sent to a large number of destinations stand to gain significantly from multicast.

In addition, multicasting offers the possibility of providing synchronization. For a set of destinations that are close together, near-simultaneous delivery of the multicast message can be expected. In any environment in which hosts attempt to synchronize their clocks, for instance, concurrent delivery to a well-defined set of receivers is difficult to simulate without a multicast facility.

Each multicast exchange involves a single communication endpoint, S, and a set of communication endpoints, R. In general, S can belong to R and receive its own distribution, though it does not have to do either. Data may flow in either direction. In the one case S is a (logically) multiplexing transmitter, in the other a multiplexing receiver. In either case what characterizes the exchange is a dialogue between a single entity and a set of entities. We refer to the single entity as the *client*, *source*, or *multicast originator* and to the set of entities as the

receivers, *servers*, or *multicast group*.

The client/server model characterizes the relationship between a multicast source and its receivers. The client has some task that it wishes to have performed; that task may be storing, manipulating, or providing data. In any case the servers (or some subset of them) will service the needs of the client based on the contents of the multicast distribution and possibly report results back to the client.

In the general case an application involves M clients and N servers. Each client multicasts its request for service to the N servers, and as many clients as possible will receive service from the pool of servers. Each individual client then participates in a straightforward one-to-many exchange with a set of servers except when the application introduces dependencies between messages with different sources in the communications system. For example, the desire for control over the order in which servers receive and process requests from different clients arises in the sophisticated crash recovery protocols for some distributed databases. These dependencies force true multipoint-to-multipoint communication.

1.1. Multicast Applications

The need for multicasting arises naturally in a number of distributed applications: resource location in a local area network (LAN) [AHAM88], distributed databases [BIRM87], conferencing [AGUI86], distributed process control [CHAN83], distributed games [BERG85], and replicated procedure calls [COOP84]. One categorization, adapted from [DEER88], of the uses of multicasting is to split them into *multi-destination delivery* and *logical querying*. Multi-destination delivery refers simply to applications in which the same information must be sent to multiple destination processes. This category includes applications such as replicated file systems and databases, teleconferencing, and distributed parallel computation. Furthermore,

data distribution that requires concurrent delivery to multiple locations, e.g. time management, fall into this category, though such applications are not widely cited in the multicast literature.

Logical querying involves locating one or more hosts whose addresses are unknown or changeable. The multicast address is a logical address identifying a group of hosts according to some logical criterion; multicast then serves as a run-time binding mechanism for associating a group identifier based on a logical grouping of processes with the actual physical servers.

Logical querying most often takes advantage of the underlying broadcast base for communication to create a location independent Medium Access Control (MAC) layer address for a set of servers. For example, a diskless workstation may use, instead of a hard-wired unicast address, a multicast address for the group of boot servers [CHER85a]. Thus, the number and location of the servers is unknown at the workstation and possibly changes with time.

From the point of view of the application domain, two different paradigms exist for multicasting. Processes as peers sharing information serves as an appropriate model for some multi-destination delivery applications, for instance; the sender and the receivers are all part of a single cluster of related processes. Any member of the cluster may wish to multicast to the rest of the group from time to time. A client/server model, on the other hand, more accurately models the relationship between sender and receivers in a logical query application, for example, since the sender (client) may be unrelated to the group of receivers to which it requests service (multicasts). In this case the multicast goes out to a possibly unknown group of servers. While the client/server(s) model subsumes its counterpart, the less general situation of a client and its servers having to belong to the same multicast group can be easier to implement and thus for a restricted class of applications might be useful.

1.2. Environments for Multicast

Three physical network environments sufficiently different to affect the nature of multicasting applications and mechanisms associated with them may be identified: multicast over a *wide-area* datagram internetwork, a *multi-segment* environment, and over a *single segment* environment. Multi-segment environment refers to one or more LANs connected by network layer or MAC layer relay nodes into a single addressing domain. A single segment environment denotes all hosts being connected to a single broadcast medium.

A wide-area environment lacks the underlying broadcast facility inherent to local networks. Consequently, routing strategies must be employed to ensure efficient multi-destination delivery over point-to-point links. Store-and-forward internetworks, moreover, have delivery times that are highly variable, difficult to control or predict, and much longer in general than local networks. Nonetheless, much work has appeared in the last few years on adding multicast facilities to datagram internets [AGUI84] [CHER85a].

Multi-segment environments also need routing services to achieve multi-destination packet delivery across segments, but each segment is, by definition in this paper, a set of nodes sharing a single physical broadcast medium. Nodes on a single segment or even separated by, for example, a small number of link-layer bridges operating at the speed of the medium will allow near-simultaneous delivery of multicast packets, but multi-segment environments may extend over a large physical area with heterogeneous segments.

In a single or multi-segment environment, we will assume that the underlying delivery mechanism consists of group addressing at the MAC layer along with a *packet filtering* mechanism at each host to effect delivery. Since standard MAC protocols including all IEEE 802 protocols, Ethernet [ETHE82], and ANSI Fiber Distributed Data Interface (FDDI) provide a large group address space and since packet filtering can in the worst case be done in each host,

these assumptions are standard ones in multicast over LANs. Of particular interest in this paper is a high throughput, multi-segment environment in which network layer routers connect a small number of LANs and real-time considerations are of importance. This environment would use the 100 Mbit/s ANSI FDDI and support redundant links between segments with automatic reconfiguration as part of the routing services.

1.3. Multicasting: A New Set of Communication Issues

As with unicasting, multicasting consists of two parts: a binding of a packet's address to some set of receiver entities and a delivery mechanism to deliver a packet to every receiver entity to which its address binds [DEER88]. But multicasting introduces a number of complexities that neither unicasting nor broadcasting has.

Multiple receivers do not accommodate address binding as easily as a single receiver. Due to modifications to the membership of multicast groups, caused by processes joining or leaving the group or failures, the same group identifier does not always bind to the same set of receiver entities. Higher level protocols are needed to manage changes in a group's status that affect addressing concerns in the communication protocol.

Delivery mechanisms are similarly complicated. In the client/server model context, one may identify for any individual multicast distribution an *ideal receiver group*. Delivery of the multicast message to possibly many subsets of the receiver group may be sufficient to accomplish the client's task; of these possibilities, delivery to some subset, the ideal receiver group, will cost the network the least with cost measured, say, as the product of bandwidth, buffer space, and processing cycles in the protocol at all nodes. Obviously, to determine the ideal receiver group for any multicast would be very difficult and costly, if not impossible. Nonetheless, we can see that in theory such a group exists.

Every receiving entity that receives and processes the multicast distribution outside of the ideal receiver group wastes system resources. This inefficiency shows up often in practice since applications may send to a receiver group that is larger than necessary due to the overhead involved in creating a new group or limitations on the number of groups. Also, in some applications, for robustness or speed, many servers are duplicating each other's efforts. A call to the group of file servers generally needs only one file server. The physical network itself will serialize the responses so that some server responds first. The work done by all the other servers on this request was (in hindsight) unnecessary.

These observations indicate two ways in which a multicast address, unlike a unicast address, can give an imprecise specification of the desired servers. Multicast facilities should be aware of the potential for variation from the ideal receiver group. The effort put into insuring reliable delivery should focus not on the entire multicast set, but on the number of servers actually required to fulfill the client's request, inasmuch as that is known.

In a multicast distribution, each member of the receiving group should ideally be able to have messages passed up to it as quickly as a single exchange between the sender and itself would allow, with the additional limitation of the degree of synchronization among group members imposed by the multicast application. That is, every member should receive at the limit of its ability within the confines of the group coordination necessary to achieve the group's task. This ideal cannot always be met in practice. This conflict represents an instance of how a multicast communication protocol must sometimes tradeoff responsiveness to individual members in order to provide the best possible responsiveness to the needs of the multicast group.

1.4. The Goals of This Research

The goals of this research are to present the issues involved in point-to-multipoint communication and to survey the literature for proposed solutions and approaches. Particular attention will be focused on the ideas and implementations that align with the requirements of our environment of interest as outlined above.

1.5. Overview

In this paper we first examine what attributes of multicast receiver groups might lead to useful classifications, what the functionality of a group management scheme should be, and how the group management module can be implemented. We then look at the services that multicasting facilities can offer, followed by a section on the mechanisms within the communications protocol that will implement these services. Finally, we identify metrics of interest when evaluating a reliable multicast facility and apply them to four transport layer protocols that incorporate reliable multicast. Conclusions drawn from this effort appear in the last section.

2. Group Management

In developing any multicasting facility one must first decide on the nature of the multicast groups and how they will be created and modified. In this section we look first at group attributes of possible interest, then at the functionality of the group management protocol, and finally at a classification of the ways to implement group management.

2.1. Types of Groups

Groups may be classified as *open* or *closed* [CHER85a]. Open groups allow any client to send to the group while closed groups only allow members of the group to be the multicast originators for the group, that is peer-to-peer communication. While open groups are more general, closed groups do model certain applications, e.g. teleconferencing, more naturally.

Having a closed group means that even for a one-time exchange, a process has to incur the delay and overhead of joining the group, the *join latency*. This operation may require notification of the new member to all other members, which for a large or dispersed group is expensive. Hence closed groups for real-time communication will not be feasible unless particular attention is paid to reducing join latency or much is known about the communication pattern of the group.

Ideally, a system will provide for both static and dynamic groups. Static, or *well-known*, groups are created at system initialization time, and their group identifiers are made available to all interested and authorized clients. Not allowing dynamically created groups severely limits the power of a multicast facility, but also eliminates management of the name space for groups.

Certain global properties of a group may be useful or necessary to specify at creation time and to record and maintain in the group management module. Some groups should have controlled access in terms of who may join them, who may send to them, or who may inquire about them. Knowing the expected lifetime of a group is useful in that it influences the amount of time and resources that can be justified in setting up routing information for the group.

Stable groups, groups whose memberships fluctuates slowly or not at all, assure a multicast originator of membership stability over the course of a multicast exchange. Failures, of course, can change the membership of a group at any time, but the notion of a stable group ensures that only failures, which are infrequent, will change the group. The idea of a stable

group, however, has yet to appear in the literature.

Another important group attribute is relative dispersion. Any set of distributed processes implicitly defines a *host group*, the set of hosts on which the processes reside. The relative dispersion of the host group within the network will determine the group's delivery characteristics at the data link layer. At least two special cases exist in which the message latency for all group members can be expected to be very similar at the data link layer. The two cases are when a multicast group resides on a single host and when a host group resides on a single segment of the network.

2.2. Functionality and Implementation of Group Management

A *group management scheme* handles the creation and modification of groups and responds to inquiries about groups. Whether the *group manager(s)* that administer the group management scheme are many or one is discussed in the next section. Group manager duties at least include how to create and destroy a group as well as add and delete members to it. Upon creation of a group, the group manager should take care of distributing all necessary information to enable multicasting to the group. In the simplest of cases, this information may be only address filters to each host in the host group, but it may also include routing information for the network, additional data structures at each group member site, and communication with other group managers. Examples of other possible duties for group managers include managing changes to and inquiries for certain global status information (e.g., current size or one of the attributes discussed in the previous section) about the group, deciding which processes can join the group, and ranking the group members based on some importance criteria.

The creation of a multicast group can have at least two different semantics. On the one hand, creating a group means that zero or more existing processes that are logically related are being bound to a single group entity identifier. In the future, processes located on any network host may be added to the group. On the other hand, the process of creation could determine on which hosts all future members must reside. At a minimum this set of hosts would include all hosts on which the initial members of the group exist. The intent then is very different in the two cases. While the former case allows more flexibility, the latter could be useful in achieving load balance and simplifying group modification procedures.

An important consideration in creating a new group is the generation of a unique identifier. Having a central agent generate all names represents a simple but unappealing solution in that the failure of the central agent causes tremendous confusion. For this reason most facilities use distributed name generation schemes. As a matter of security, name generation schemes where the possession of a few names allows the user to figure out the naming algorithm and thereby possibly send to or inquire about restricted groups should be avoided [GAIT89].

Often random or pseudorandom bit patterns are used to generate group names. Given long (32 or more bits) identifiers, the sparseness of the names in the name space should prevent any duplication and, in addition, provide security from any guessing of names by unscrupulous users [GAIT89]. In his MCL system Hughes suggests the method of concatenating the host identifier and the time of day to create unique group identifiers. This method takes advantage of the fact that in any system the host identifier should be unique and time is assumed to be monotonically increasing [HUGH87].

In the V distributed kernel [CHER85b], a new group identifier contains a random bit pattern, part of which largely determines the host group for the new multicast group. Two

aspects of this scheme are worthy of note. First, this random assignment of host groups will result in hosts belonging to groups, but not having any members of that group resident, and thus hosts accept delivery of packets for which there are no receivers on that host.

As noted in [CHER85b], there is a trade-off between the number of host groups and redundant messages. If, at the one extreme, one process group per host group is used, each host only receives packets for which there are receivers on that host. But a larger name space for all the possible host groups is required. At the other extreme, all process groups can share a single logical host. An early implementation of the V system on an Ethernet used this solution, broadcasting all packet traffic to every host in that V domain.

Second, encoding the logical host group into the group identifier implies that modification to the group membership will require the issuing of a new identifier for the group, barring the crude solution of having a single host group mentioned above. Cheriton does not explain how this problem is taken care of in the 10 Mbit/s Ethernet implementation of the V kernel, but a paper describing a much more recent implementation of the V kernel states explicitly that V process group identifiers are stable [CHER89].

Assuming that a group has a single group manager, the addition of a member to the group requires only a unicast of the group address to the new receiver. (If adding a group to a group is allowed, then a multicast to the new members is in order.) Deletion of a member may require no communication at all if one may assume that whatever state information on the multicast exchange that exists at the receiver will be automatically discarded in time, for example by caching strategies and connection time-outs. In general, not cleaning up at the receiver introduces the risk of delivery of an old message to a group member or a message in a group reusing an identifier before that identifier has been erased at a member of the group that previously used the same identifier. The results of such a delivery will be unpredictable.

Finally, the group management protocol should automatically deallocate the group identifier in the special case of the last member of a group leaving.

2.3. Distributed vs. Centralized Management

Paliwoda identifies two basic ways of implementing the group management facility [PALI88]. First, a single *agent*, or *manager*, external to the group may coordinate group changes and handle the other functions of such a facility. In this case the receivers are aware of no group state information except the group address. Typically this situation results in a client of a group, a process coresident with a client , or a manager that holds all state information for all multicast sets in the manager's domain acting as the group's agent.

A single manager can be assisted by having *local agents* at each client site. Local agents make available groups' state information to clients on its host. The local agents either periodically poll the manager for the updated status of the local agents' groups, or the manager multicasts changes as they occur.

A central manager, however, represents a single point of failure. This leads to the idea of having a number of managers that share information about groups. A replicated and distributed pool of group information brings with it the problem of maintaining a consistent global view of each group. Managers must communicate changes in groups' statuses. This problem suggests having a super manager to coordinate manager communication.

The concern over having a centralized agent for each group should be proportional to the size of the network and to the number of clients multicasting to the group. In a wide-area environment a single agent per group may mean long response times when querying the remote agent. Recreating a large group may be a very lengthy, expensive task. Not surprisingly, in their proposal to provide a multicast facility for the DoD Internet [CHER85a], Deering and Cheriton avoid a centralized agent and propose a scheme where hosts are agents for local

processes and gateways agents for host groups.

Self-administered groups represent another possibility. Instead of maintaining group state information at a client's site, local agents exist at each group member's site. The members of a group have the power to coordinate group changes through communication with other members. Unlike in the previous set-up, each group member has a copy of the group status information available locally and the ability to access it. Prospective clients outside the group query a member's local agent to inquire about a group's state information, including the group identifier. Self-administered closed groups guarantee that every multicast originator has the group state locally available, instead of having to find and query a remote agent as in the other administrative model.

Birman [BIRM87] provides an example of self-administered groups. A special group multicast primitive, GBCAST, allows group modifications to be atomic and ordered with respect to all other transactions as well. Thus, every group member has an accurate record of the group's status and can act on it, including providing status information to current and potential clients, without further agreement protocols.

3. Multicast Services

This section deals with the semantics of the user services that a multicast facility should offer. A user process makes calls to the *multicast kernel*, an application layer protocol sitting between the client and the transport layer data transfer protocol, using a set of primitives. These primitives provide services that fall into two categories: group manipulation and query and multicast data transfer.

3.1. Group Management and Query

Through one set of primitives, the user should have the ability to create and manipulate group membership as well as query for useful status information on a group. The exact semantics for this interface, however, depends heavily on how the group management facilities have been implemented. Querying for group state information can involve either a local agent or a remote manager. The group attributes of which the group management scheme might keep track were discussed in section 2.1.

The information provided to a client about a group must be considered a hint to the client about the current status of the group. Without some expensive overhead, there can be no guarantee that group membership is stable over the course of a particular multicast exchange. Possible solutions to this problem include locking membership and using a two-phase commit algorithm for delivery or allowing only one client per multicast group. The former is unacceptable in a real-time environment, and the latter causes an explosion in the number of multicast groups while still failing to deal with shrinking membership due to failures. In general, however, not knowing the status and membership of a group precisely is an acceptable situation, with the notable exception of a client wishing to send a distribution that must reliably reach every member of the group.

Given multiple clients per multicast group, the issue arises of how to divide the responsibility between the two sides of this interface for group modifications. The group management scheme must first decide if a client has the right to modify a group. A simple solution is to assume that having the name (group identifier) of a group constitutes a capability to perform group operations on that group; more secure schemes would require some sort of key to be presented by authorized group modifiers. How the group management scheme coordinates concurrent changes to a group and propagates the updates to interested parties

depends on the implementation with either a central group manager or a super manager in the key role.

3.2. Multicast Data Transfer

The multicast client should be able to control the reliability of a multicast and convey to the data transfer protocol the response behaviour expected. Towards the first goal, the send-data primitive should allow the user to specify the number of receivers in the multicast group that must reliably receive the distribution. This is called *k-reliable* multicast, where *k* is an integer between zero (inclusive) and the number of members in a multicast group [CHER85b]. Moreover, the sender should ideally be able to pass a list of any specific group members that must see the distribution. Specifying the reliability desired allows the user to convey to the communication system the nature of the service desired and the communication system should take advantage of this information for whatever efficiencies can be gained from it.

Symmetrically, if the sender expects responses from the multicast group members, the sender should specify the response behaviour expected for the benefit of the communications system. How many responses are required can either be explicitly passed as a parameter or the underlying system instructed to continue to make responses available as they arrive until told to stop. The former method, if the client knows this, is preferable in that it allows the communication protocol to issue an abort message to remote servers working on an old request more quickly than if this function is left up to the client process. Again, the multicast originator may wish to specify specific servers from which responses are required.

Another useful parameter for a *receive-data* primitive may be a *response filter*, an algorithm that will allow the transport layer protocol to decide whether a response is useful or valid. Often this decision can be made without having to buffer the entire response, and this mechanism can reduce the delay in issuing an abort request message to remote servers once all

necessary responses have been collected. Also, in the case of a single response expected at the application layer, conflicting responses at the transport layer can be voted on and agreement reached as to the "correct" response.

The most difficult multicast transfer to manage is one in which all members of the group must reliably receive the distribution, an *all-reliable* exchange. In this case, the sending site must know explicitly the membership in order to ensure that all members are reached and, if desired, all members send responses. Handling changes in group membership during an all-reliable exchange can complicate the semantics of the protocol considerably. An alternative is to take a snapshot of group membership, a *version*, and use that for the duration of the exchange. A parameter in the send-data primitive for the version number allows the client to link a query about a group to a particular multicast distribution to that group. This allows the client process to know, upon successful completion of the all-reliable multicast, explicitly which processes received the distribution, even if the current membership of the group is different.

The strategy mentioned above takes away the transparency of a multicast group. If a client is not so knowledgeable about the members of a group, but only wants to be sure, for example, that the group of all name servers or multicast group managers receives an update, then control over all-reliable delivery should be left to the transport layer protocol. As long as the semantics of all-reliable delivery are clearly laid out, the transport layer can devise its own strategies for handling shifting membership. For example, given a self-administered group, each receiver in an all-reliable distribution could report its membership version in a response message upon receiving the multicast data. If all report the same version, the sender assumes success; if not, the client site can take appropriate action.

Members joining a group during an exchange of any reliability, once known about, can be accommodated easily enough. The loss of members due to failure or deletion, however, can cause a reliable exchange to be impossible to complete. In addition to whatever timeout mechanisms exist at lower layers, the application process should have a field in the send-data and receive-data primitives to allow it to time out an exchange. Such a field is dangerous in that it requires the user to make guesses about the duration of events that it knows little about, but the field is also useful in that an application knows how long it can afford to wait for responses or confirmation of delivery. In cases where the application has no definite constraints, the timeout mechanisms at the lower layers can be relied upon by setting the parameter to a value representing infinity.

Finally, a send-data primitive may include a parameter of, say, a bit-mask that selects from a menu of special services that the multicast protocol can offer. One useful piece of information to convey to the underlying message service is whether the application process intends to send more than one message to this group, that is whether a connection-oriented or connectionless service is being sought. The user, of course, has no guarantee of how or if this information will be used, but efficient use of the communication system is the goal. This parameter could also serve to expose powerful features embedded at lower layers. VMTP, for example, proposes to have a conditional delivery option wherein a multicast message is only delivered if the receiver can immediately process it [CHER89]. A real-time application might set this option on or off on an exchange-by-exchange basis.

4. Mechanisms

In this section we present mechanisms that are used to support multicast services in a transport layer protocol. We look first at the mechanisms for the data link and network layers

and then move up the protocol stack.

4.1. MAC Layer Addressing and Packet Filtering

At the Medium Access Control layer, a true multicast facility must be capable of addressing a single frame to multiple destinations. Most current MAC layer protocols support a wide group address space. The 10 Mbit/s Ethernet [ETHE82] reserves the most significant bit to indicate a group address and provides the remaining 47 bits to create 2^{47} unique group addresses. The ANSI FDDI standard is considering an addressing structure similar to the IEEE standard discussed below.

Similarly, the IEEE 802.5 Token Ring Standard, using 48 bit addresses, provides 2^{46} group addresses. Setting the most significant bit in the address field indicates a group address; the second most significant bit indicates whether the address is locally or globally administered. The remaining bits constitute the address space. All bits set indicate a broadcast address for all active stations on the ring; all other addresses designate a group defined at configuration time or by a higher-layer convention. The exact nature of the locally administered 46 bit address is not defined in the standard, but a hierarchical structure with one field of 14 bits for the ring number and another field of the remaining 32 bits for the station number is recommended.

Either bit-significant or conventional encoding can be used in both the station and ring fields. In [STAL87] two advantages of bit-significant addressing are pointed out. First, a station can store its membership in a simple bit-significant mask so that matching an incoming group address involves only a logical AND operation. Second, a packet may be delivered to multiple groups. These advantages come at the expense of greatly narrowing the multicast address space.

In addition to the ability to create MAC layer group addresses, a LAN-based multicast facility requires that all hosts can recognize which multicast packets are intended for them. This *packet filtering* ideally should be done entirely in the network interface hardware since doing it in software is orders of magnitude slower. Many current network interfaces, however, only allow a small number of group addresses to be associated with any one station. To take two Ethernet interfaces as examples, the Digital UNIBUS Network Adapter, DEUNA [DEUN83], supports only ten while the Intel 82586 LAN Coprocessor hashes incoming group identifiers into 64 address buckets [MICR86].

4.2. Multi-destination Delivery: Routing Services

In our environment of interest, separate LANs are connected into a single addressing domain by routing services. Unless group members are all located on the same physical segment, a multicast transmission uses the routing services of the network. A useful model for routing in any packet-switched network is an undirected graph. The nodes of the graph represent either point-to-point links as with gateways in a datagram internet or multi-access links as with transport layer routers or link-layer bridges connecting LANs and the segments within LANs. The edges of the graph then represent the physical connections between nodes.

Routing algorithms determine over which links a packet travels as it traverses the network from the packet's source node to its destination node. At each node the *routing element*, whether gateway, router, or bridge, receives an incoming packet and routes it out zero or more outgoing links based on the routing algorithm in use.

Following [FRAN85], we identify below appropriate metrics and evaluate classes of routing algorithms, coupled with their associated addressing techniques, for multi-destination delivery.

4.2.1. Metrics for Routing Strategies

A routing algorithm must be evaluated in how it performs for any one multicast packet and how it performs over the course of a multicast session, that is, the lifetime of a connection between some client and its multicast receivers. For a single multicast distribution, consideration should be given to the amount of bandwidth consumed, the delivery delay from sender to receivers, and the amount of state information and computation necessary for delivery. For a multicast session, relevant metrics include the speed and cost of preparation, maintenance, and failure recovery. Finally, as the number of members in a group grows, the performance of a routing technique should scale in a linear fashion.

The initial cost of setting up the multicast session includes distributing multicast information to all members and the appropriate routing elements. Possibly a long set-up procedure may create structures that lower the average cost of individual multicast distributions. In the case of statically created multicast groups, all routing set-up can be done at system initialization time. Maintenance of routing information may be necessary for any group that allows modifications to its membership. Triggered by either the loss of either a routing element or an actual channel, failure recovery involves an adaptive scheme to find a new route, which, given real-time considerations, must be done quickly.

4.2.2. Survey of Classes of Routing Algorithms

4.2.2.1. Flooding

The brute force method for multicasting is to broadcast identical packet copies to *all* hosts in the network. A routing element simply copies the incoming packet onto every link to which it is connected except the one on which the packet arrived. Hosts in the multicast group will receive a copy of the packet while the network interfaces at all other hosts will discard their copies of the packet. In order to avoid endless duplication, given that redundant links between

the segments exists, when using flooding, some mechanism, typically a hopcount or duplicate detection in the routing element, to limit the lifetime of packets must be available.

Flooding wastes large amounts of bandwidth, placing packets on many parts of the network where there are no group members and often delivering more than one copy of the distribution to the same segment. Loops in the topology can cause an avalanche effect. But flooding does have some attractive properties. Since all paths are explored in parallel, flooding is guaranteed to find the shortest path to each receiver. Flooding is very robust, requires no maintenance, and requires no state information to be kept in the routing elements. Nonetheless, outside of a small network with very few redundant links or some set of extraordinary circumstances (e.g., robustness of a single multicast distribution at a great penalty to overall network performance), flooding is almost never a viable alternative.

4.2.2.2. Separate Addressing

Completeness compels the mention of achieving multicast by performing a series of unicast, that is, send separately addressed packets to each destination. This technique makes little sense over broadcast LANs with packet filtering support.

4.2.2.3. Multi-destination Addressing

In this scheme the sending host sends out a few (possibly one) multiply addressed packets, each of which contains a subset of the addresses of the members of the multicast group. When a packet arrives at a routing element, multiple copies of the packet are constructed, one for each channel connected to the routing element that leads to a destination. The packet leaving on each channel contains the subset of address destinations that can be reached by that channel. For every destination then there will eventually be an out-bound packet from some routing element with only that destination's address; this packet is then treated like an ordinary unicast.

Multi-destination addressing only makes sense when routing elements are capable of originating packets; bridges will not do. It also implies the use of a variable number of addresses in the headers of packets and thus a great deal of parsing at the routing elements. This burden on routing elements is undesirable in a real-time environment. The Metanet project, which seeks the interoperation of several networks for the US Navy, proposes multi-destination addressing as an option of the DoD Internet Protocol with only minor changes to IP while preserving interoperability with IP modules not supporting multicast [AGUI84].

4.2.2.4. Partite Addressing

This method combines separate and multi-destination addressing. The network is logically partitioned for addressing purposes. A multicasting host sends one copy of the distribution packet to each of the logical partitions, which could correspond to subnets or channels, and from there the packet is delivered to the receiving hosts, using whatever methods are available in that particular partition. This method is suitable for multicasting between a number of broadcast segments, but it fails to take full advantage of the underlying broadcast medium.

4.2.2.5. Single- and Multiple-tree Forwarding

By far the richest category of routing approaches uses various types of trees constructed from the network graph. A multicast packet is directed through the network based on the local image of a tree structure at each routing element. Spanning trees are natural structures to consider in attacking 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. If the edges are weighted with a cost, then the cost of a spanning tree is the sum of the costs of all its edges. A minimum spanning tree (MST) then is a spanning tree whose cost is as small as

possible over all spanning trees of the graph. A minimum spanning tree thus minimizes the cost to the network as a whole of a packet transmission.

If the edges of the graph are weighted according to delay, a shortest path tree for node *N* is one with the shortest possible path from *N* to any other node. That is, one takes the shortest paths from *N* to all the other vertices and combines these edges to form a tree. This tree may not be unique and there may be combination of paths that would form cycles if we tried to build a tree that included them; nonetheless, there is always at least one such tree for each node in the graph [WALL80]. Unlike a MST, a shortest path tree is relative to a particular node in the graph and that particular node becomes the root of the tree.

A minimum spanning tree can be used to minimize the number of packets generated in forwarding, but delay depends on the position of the receiver node within the spanning tree relative to the source node [DALA78]. In contrast, shortest path tree routing minimizes delay, but possibly not cost. *Source-based forwarding*, *reverse path forwarding* [DALA78], and *center-based forwarding* [WALL80] represent three important delay-minimizing techniques that all make explicit or implicit use of the shortest path tree for some given vertex in the network graph. Source-based forwarding would involve storing at every node in the network a local image of the shortest path tree for every other node in the network. On a given multicast, a node could determine which tree should be used by looking at the source of the distribution. While this technique would minimize distribution time and bandwidth, it in general requires too much information in the routing elements to be practical. Reverse path forwarding simulates multiple trees without actually maintaining them by using routing tables and two additional lists. Center-based forwarding uses for all routing a shortest path tree for a particular node that, in some sense, is in the center of the network and thus generally produce routes with low, but not always minimum, delay.

It should be noted that minimizing delay and minimizing cost involves more than just labeling edges of the graph differently. For example, consider a multicast from the center of the network out in two directions, conceptually east and west. The total cost of the multicast will be the sum of the cost incurred in the east direction and the west direction, but the total delay will be the maximum of the two delays [WALL80].

4.2.3. Summary

In the delivery of packets on a broadcast internet, the central problems may be summarized as (1) how to find the receivers and (2) when to split the original copy of the multicast message into the copies that will arrive at the receivers. For (1) the possibilities lie in the spectrum between not finding the receivers (e.g. flooding) and having all of the destination addresses before transmission (separate addressing). As to (2) the minimum possible number of packets generated is achieved by using a channel-based spanning tree coupled with the principle of multi-destination addressing; that is, each destination, D, shares a single copy of the multicast packet for as long as possible with other destinations whose shortest delivery paths have branches in common with D. Basing the tree on channels avoids delivering multiple copies of the packet to the same broadcast medium. Flooding will in general produce the most packet copies during a logical multicast, though an estimate for the bound on the number of packets produced by flooding (or selective flooding) may be not easy to compute. In a crude sense the trade-off in multicast packet delivery comes down to bandwidth versus complexity and cost; the more effort to pinpoint the set of receivers for which the multicast is intended, the less wasted bandwidth in generating duplicate or useless copies of the message.

4.2.4. MAC Layer Bridges in an Arbitrary Topology

In our environment of interest, the routing algorithm for a multicasting scheme must take into account the fact that redundant links between segments represent the desire for robustness in the face of link failures. *Transparent bridges*, bridges as defined by the IEEE 802.1 Medium Access Control Bridge Standard [IEEE85], use a forwarding scheme dependent on the existence of a single path between any pair of nodes. Hence the standard approach applies a distributed algorithm to transform the arbitrary mesh topology of the given network into a single, acyclic spanning tree. Topology changes are detected by intra-bridge communication, and a new spanning tree determined.

In [SINC88] the authors develop algorithms that allow the use of bridges in extended LANs of arbitrary topology without confining the traffic to a single spanning tree. The procedure is to decompose the graph into some number of spanning trees, number them, and then mark each packet as traveling on a single tree. Then the basic technique used by transparent bridges of building their routing tables based on the source address of packets passing the bridge [BACK88] 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 straightforward routing algorithm for multicasting. It does depend on two-way communication to resolve a path so that hosts involved in the multicast cannot be passive listeners or, equivalently, 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. 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. All the preparation cost of determining and numbering a set of spanning trees can be confined to

network initialization time. Maintaining multiple tree forwarding information in the bridges is a major cost, but not unthinkable given the ever falling cost of memory. The most devastating blow to the usefulness of this idea results from the fact that indicating a tree number in a packet's address is not provided for in standard MAC layer protocols.

In [DEER88] extensions of the standard bridge routing algorithm to accommodate multicast are presented. The article describes how routing tables can be augmented to handle multicast addresses and a scheme whereby the hosts on which multicast group, G, has members issues periodic *membership report* packets by which bridges learn the link(s) onto 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.

4.2.5. Network Layer Routers and Efficient Multicasting

The bridge routing ideas proposed in [DEER88] could be applied to routing between LANs via network layer routers, if a single spanning tree across all the links were used. In thinking about multicast routing in a wide-area environment, the authors consider and reject this approach as unacceptable due to the fact that a single spanning tree is relative to a specific source and a specific multicast group. Thus, while Wall's center-based forwarding is a possibility, they believe that a shortest path tree should be required for delivery of a multicast packet.

Subsequently, the authors observe that every shortest path multicast tree rooted at a given sender is a subtree of a single shortest path broadcast tree rooted at that sender. The latter tree then may serve as a basis for routing when properly pruned. Four schemes with increasingly

precise pruning of the broadcast tree, along with a correspondingly increasing amount of routing overhead, are presented, all based on reverse path forwarding [DALA78] and assuming that the routers use *distance-vector* routing for unicast. In distance-vector routing, also known as the Ford-Fulkerson or Bellman-Ford algorithm, each router stores a vector containing a destination, a distance to that destination (e.g., hop count), a next-hop-address for the next router (or routing element), a next-hop-link for the outgoing link, and an age field to time out obsolete paths.

The first method presented prunes the broadcast tree by performing reverse path forwarding on all packets with a multicast destination address. That is, a router looks at a multicast packet, determines if it arrived along a path that represents the shortest path back to the source, and if it does then forwards it on all outgoing links except the one on which it arrived. This performs the multicast, albeit with a large number of duplicate packets. These duplicates appear not only because of the flooding of the packet on all outgoing links, but also since more than one router on a multi-access link will forward a copy of the packet onto that link.

In a more sophisticated approach, a *parent* router is chosen for each link, relative to each possible source, S. The parent router is the router with the shortest path to S where, in the case of more than one such router, an arbitrary winner based on the lowest address is chosen from the set of possible parents. Over each of its links, a particular router learns each neighbor's distance to every source since this information is conveyed in the routing tables. Hence, each router can decide independently on whether it is the parent of a particular link, relative to each S [DEER88]. An extra field, *children*, is added to the routing tables. With the election of a single parent per segment, only the parent elects its segment as a child link during a particular multicast, significantly reducing the number of duplicates.

The authors go on to lay out more complicated schemes to increase the "precision" of the multicast delivery further. They also explore multicast extensions to another popular unicast routing algorithm in datagram internets, *link-state* routing. This technique is very expensive, requiring all routers to acquire knowledge of the complete topology of the network and independently compute the shortest path spanning tree rooted at themselves. Given that the authors' domain of interest in this article, datagram internets, and ours is quite different, these more complex techniques are completely inappropriate for our purposes.

4.3. End-to-End Control Mechanisms

At the transport layer reside the mechanisms for providing reliability as well as techniques for specialized services such as filtering responses or ensuring delivery orderings.

4.3.1. Filtering Responses

As an optimization, a multicast application that expects responses to its request may supply the protocol with a filtering algorithm that allows the protocol to decide which responses to pass up to the application and which to discard. This filtering can significantly improve efficiency since the decision about the usefulness of a response can usually be performed before the entire response is buffered. Moreover, by determining as soon as possible when all the desired responses have been received, the filtering mechanism allows an abort message to be sent to all the servers as quickly as possible in order to save network and remote host resources that are wasted in working on old requests.

For example, responses to a request for data replicated at several servers could contain a timestamp sent in the first data packet with which the filter algorithm compares another timestamp. If the data is too old, it is not buffered and an abort message is sent to the server sending the old data. The filtering algorithm, however, does not have to involve explicit

manipulation of the response data. In locating a resource in a LAN, the implicit filtering algorithm is often that the first response is taken. In this case the physical network has been used to serialize responses, and the protocol uses that ordering to determine which response to make available to the client.

Another form filtering takes is some technique by which some subset of a multicast group is told to respond to a request. That is, the best way to eliminate unwanted responses is to not have them generated in the first place. A mechanism for doing this could be as simple as having the group management protocol number the group members. For a distribution requiring five responses then, the multicast request packet contains a field with the integer five in it. Group members examine this field upon receiving the request and only respond if they are one of the first five members in the ordered group.

Ordering members, or any method of restricting a request to a subset of the group members, assumes that the membership of the group is explicitly stored in the group manager. Moreover, it requires a good deal of overhead in the maintenance of groups since every group modification forces a reordering. Finally, it is unlikely that any single ordering will serve the needs of all the clients of a multicast group. For a stable group in which members provide identical services (e.g., distributed computation on processors of similar power or servers holding replicated data), a simple ordering scheme might however prove worthwhile.

4.3.2. Reliable Multicasting

In the ISO OSI Reference Model [ISO7498] the transport layer classically handles reliable and transparent data transfer. Toward this end, acknowledgement and retransmission schemes ensure that lost packets are resent. Flow and rate control parameters are used to adjust the volume and rate of the data stream to the capacities of the receivers. Reliable multicasting

however, forces a rethinking of the strategies generally used for reliable unicasts.

4.3.2.1. Acknowledgement Strategies

In order to perform reliable multicasting the sender must have some assurance that each receiver indeed received the multicast text. Acknowledgement messages (ACKs) may be sent back to the sender either when messages are received, *positive acknowledgements*, or when they are not, *negative acknowledgements*. That is to say that acknowledgement strategies either place the burden of ensuring reliability primarily on the sender or primarily on the receivers.

Metaphorically, the latter may be characterized as *publishing*. Publishing requires that information sent to a group, the *subscribers*, be filtered through the *publisher*, which collates and numbers the information before issuing it to the subscribers. Upon noticing a missing issue by a gap in the issue numbers or a new issue not arriving in the expected time, a subscriber requests the *back issue* from the publisher, that is a negative acknowledgement is sent. A slight variant on publishing appears in [CHAN84] since the role of the publisher, instead of being played by the sender, is passed around from receiver to receiver.

Putting the burden of reliability on the sender (as positive acknowledgement schemes do) means simply that the sender resends the message to the group until it receives replies from all members of the group. An obvious optimization is to resend the message only to those processes that have failed to reply, but for this the sender must know the membership of the group. Since some receivers may receive the distribution multiple times, a useful feature may be to have a user specified switch as to whether requests should be reexecuted in the case of retransmissions (e.g., if a result changes with time) or saved copies of the initial response resent. Also, complications may arise from the fact that connection management and path maintenance in the network can be dependent on a certain regularity of traffic between sender and receiver.

Either of these two general methods can choose to have acknowledgements unicast or multicast. A receiver multicasting its acknowledgements allows other receivers to overhear the transmission, and they can then use the information in the acknowledgement to learn about the state of fellow group members. Also, since receivers can avoid generating a duplicate request, this technique insures that a large number of simultaneous responses do not overrun the sender.

Unicasting acknowledgements is expensive in terms of bandwidth and the complexity at the sender in managing multiple windows. Large groups may overwhelm the sender or some intermediate bottleneck (e.g., a routing element) with acknowledgements that, given simultaneous delivery to the receivers, will tend to come in bursts. A major advantage in unicasting acknowledgements is that, from the communications protocol's viewpoint, multicast and unicast become almost identical at the receiver.

4.3.2.2. Retransmission Strategies

Multicast is more sensitive to errors than unicast in the following way. Consider that for a reliable unicast transmission with positive acknowledgements, a retransmission costs one extra packet event since the original transmission is lost and never reaches its destination. For a multicast retransmission, assuming unicast positive acknowledgements, $1+3*R_{first}$ additional packet events are required where R_{first} is the number of receivers that received the original transmission. 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, which then possibly shares the same medium.

A number of approaches can be used to combat the problem of the high cost of retransmissions due to errors in multicasting. On the assumption that a packet lost at one receiver is likely to be lost at a nearby receiver, multicast groups could be chosen such that they can be partitioned into subgroups that are homogeneous in terms of physical closeness or response time. In this way retransmissions to selective subgroups could reduce the cost to the entire group of individual delivery failures. In other words this hierarchical structure for each group would provide a "firewall" for retransmissions. This strategy is limited primarily by the tolerance of the application for variance in delivery times to different members of the multicast group and by the overhead of such a selective retransmission scheme.

Another approach is to keep a list of group members that have not replied and after a timeout to unicast retransmissions to the failing receivers. This scheme assumes that the sender has knowledge of all the members of the group and, even more, has the explicit address of all members. In the general case this is not true and in fact this condition severely limits the applicability of the scheme. One notable exception might be well-known groups. In [CROW88], as an optimization the protocol either multicasts retransmissions or unicasts them depending on whether or not the proportion of failed deliveries to group size is larger than some threshold value.

The use of a threshold value could be inaccurate in predicting the true needs of retransmission policy in the following way. Suppose some large number of receivers failed to acknowledge the multicast distribution; the threshold value having been surpassed, the sender re-multicasts the message to the entire group. But if all of the failing receivers were on the same channel or even same small number of channels, it is clear that a single unicast distribution to that channel could rectify the problem. Hierarchical subgroups are useful in this regard.

4.3.2.3. Flow and Rate Control Mechanisms

A fundamental problem of having multiple receivers for a reliable message is that communication between the sender and receivers becomes limited by the weakest member of the group of receivers. Flow and rate control for the group are forced to restrict the data flow so as not to overwhelm the "weakest" receiver, the receiver with the most limited reception capacity; otherwise that receiver can potentially degrade service badly by either missing the distribution or, if the transfer is reliable, missing the distribution and causing costly retransmissions. If groups contain relatively homogeneous members, this problem is small, but a single weak receiver amid a group of powerful receivers results in a very inefficient data transfer.

4.3.3. Delivery Orderings

For some applications the relative delivery orderings of messages sent from multiple sources must be controlled at every receiver common to the sources. The strongest possible constraint is that there be a global ordering on all messages in the communications system. That is, if A and B are both multicasting two messages to C, D, and E, then C, D, and E will have exactly the same message reception orderings. If C receives in order A1, B1, B2, and A2, then D and E must also receive in order A1, B1, B2, and A2. A more relaxed condition is *FIFO ordering* in which each receiver receives the messages of each sender in the order in which the sender sent them. Thus, in our example, the FIFO ordering is upheld if C receives its messages in order A1, B1, B2, and A2 whereas D gets them in order A1, A2, B1, and B2. There exists an ordering stronger than FIFO but weaker than a global ordering called *causal ordering* in which delivery order is enforced when desired, but with minimal synchronization.

Generally the issue of message delivery order arises from concerns about maintaining consistency in data updates and fault tolerance in distributed processing applications. To

provide strong ordering, multiple phase handshakes may be required and in general the more strict the ordering requirements, the more synchronization delay required to enforce them. In [SCHM88] Schmuck tackles the problem of the trade-off between the degree of message delivery ordering provided by a reliable broadcast (multicast) protocol and the cost to implement this ordering guarantee. Techniques for constructing efficient asynchronous solutions for applications that do not require global ordering are developed, though it is also proven that whether a problem (application) must have the global ordering property is undecidable.

Birman and Joseph [BIRM87] have shown from their experience with supporting fault tolerant groups in the ISIS project at Cornell that causal ordering is at the same time strong enough for many applications and yet imposes an ordering sufficiently weak, particularly in comparison to a global ordering, to allow processes to take much advantage of concurrency and hence improve performance. In order to ensure correct delivery orderings, however, elaborate queue management must be performed. In the ISIS system, entire buffers of previously delivered messages are sometimes sent from one node to another in order for the receiving node to understand fully its message history. Consequently, delivery ordering mechanisms in general are outside the scope of a real-time communications system.

5. Evaluation of Reliable Multicast Data Transfer Protocols

In this section we consider the key issues in attempting comparisons between reliable multicast transport layer protocols (RMTPs). We believe these are:

- potential for real-time
- limitations on groups

- compatibility with unicast
- cost
- reliability
- special services

Comparison of protocols in general is a notoriously difficult problem. No two protocols have the same set of features nor do they achieve the same functionality. Modeling dynamic behaviour is tricky and dependent upon the communication environment. What we can hope to do, however, is to identify categories of functionality that must be provided in order for reliable multicast transfers to be handled efficiently and to give the widest possible range of services to higher layers. While only gross relative measurements and not numerical results are generally the extent of our measurements, these metrics serve two important purposes. First, they provide a checklist of the aspects of a given reliable multicast protocol that should be inspected in evaluating the protocol. Second, they give some idea of where in the spectrum of possibilities the RMTP in question sits.

Real-time Potential

RMTPs can be divided fairly easily into those that can be potentially used in a real-time environment and those that cannot. For example, a protocol such as [CHAN84] that forces the members of a multicast group to enforce a global delivery order and to come to distributed agreement during error recovery is inherently prone to long and highly variable message latencies that are unacceptable in a real-time environment.

Limitations on Groups

A multicast transfer protocol may optimize its performance in various ways using implicit assumptions about the multicast groups in use. Various timing dependencies can be

reasonably estimated if group members experience nearly simultaneous delivery; if such timing assumptions are exploited in the RMTP itself, multicast groups for the RMTP must all be relatively close together and network layer routers between members is excluded. The tightness of the timing dependencies determines the extent to which multicast groups are restricted.

RMTPs may implement either open or closed groups. Open groups are more general and hence more difficult to implement for efficient data transfer. If only closed groups are allowed, however, the group management protocol must deal with the problem of applications that are sensitive to the join latency.

Compatibility

A multicast scheme should follow the form and usage of unicast transmissions. Interest in multicasting is high and growing as its value has become widely recognized [PALI88], but the caveat to be heeded is that multicast is rare in comparison to unicast. Hence, even the slightest overhead imposed on common packet processing by adding a multicast capability should be carefully scrutinized. Multicast can add overhead in two ways: (1) by reducing overall throughput by forcing checks for options during common processing, and (2) by adding a great deal of code to the protocol's implementation. The latter becomes a crucial issue as we move toward the time when lightweight transport layer protocols can be implemented in hardware. Thus, inasmuch as possible, a RMTP should utilize the mechanisms of the unicast protocol in which it is embedded. Mechanisms such as those for flow and rate control, connection set-up and teardown, and routing services should be shared.

Cost

The cost of a multicast exchange under a RMTP refers to the usage of critical network resources, namely processing cycles, buffer space, and bandwidth. An indication of the amount of processing needed for an exchange can be seen in the number of timers needed, the number and intricacy of supporting data structures for multicast data transfer, and the degree of buffer management. Buffer space and bandwidth usage can in general be traded off against each other.

Reliability

A crucial point about reliability in multicast exchanges is that only the application can know what degree of reliability is really needed. That is, partial delivery success is often acceptable, but the application must have a way to convey this information to the RMTP. Hence, the degree of flexibility in the user interface in specifying the reliability sought is especially important. For the protocol to provide more reliability than needed will be wasteful of communication subsystem resources; to provide less than needed violates the integrity of the system. In any case, a RMTP must provide, at a minimum, reliable delivery to multiple destinations. Reliable transfer in the most general case includes the detection of receiver failure during the exchange, but an alternative is to guarantee to reach all active receivers.

Special Services

Finally, multicast has properties that unicast does not and thus may be used in applications that require special services. Exploitation of the concurrency found in multicasting could prove very valuable in a real-time environment, and a RMTP can provide the primitive mechanisms with which to tap this power. This aspect of multicasting will certainly come to the forefront as time management becomes more prevalent. The value of special services can only be determined with experience, but, if they can be included with low

overhead, their presence is a plus in that, if nothing else, it encourages experimentation by application programmers.

6. Four Data Points

To assess the state-of-the-art for reliable muticast transport protocols, we will look at four examples from the literature: the Xpress Transfer Protocol (XTP), the Versatile Message Transport Protocol (VMTP), an experimental protocol (referred to below as CP) proposed by Crowcroft and Paliwoda [CROW88], and an experimental protocol based on Negative Acknowledgement with Periodic Polling (NAPP) in [RAMA87]. These four examples represent detailed attempts to build transport layer reliable multicast protocols. They do so in a way that makes them candidates to be used in a real-time environment. The protocols examined can be loosely described as intended for general purpose, though each is in fact tied to specific paradigms, assumptions about environment, and some set of design goals. All share the assumption of working over high bandwidth networks with low error rates. The exact assumptions about environment will be pointed out as each protocol is introduced. Each protocol will be examined in light of the discussion in section 5. Table 1 at the end of the section summarizes the features and characteristics compared below.

In order to lend some concreteness to the discussion, three simple scenarios for message transmission will be examined when useful. In each scenario a 4-packet message is to be sent from source, S, to a multicast group of three members, R1, R2, and R3. In *scenario 1*, the four packets are sent and correctly received in order at all group members. In *scenario 2*, all receivers receive the first two packets, followed by the fourth packet, but fail to receive packet 3. Error control mechanisms are invoked and lead to retransmission of packet 3 in a best case fashion. Finally, *scenario 3* is the same as *scenario 2* except that a single error control packet is

assumed to be lost in a worst case fashion.

6.1. XTP

The Xpress Transfer Protocol, XTP, is a lightweight transfer layer (the transfer layer being defined as the transport and network layers merged) protocol being developed by Protocol Engines, Inc. It is designed to provide the end-to-end data transmission rates demanded in high speed networks without compromising functionality, including in particular, support for reliable multicast. A major goal of the XTP effort is to keep XTP small enough so that it can be implemented in hardware as a VLSI chip set. The discussion below examines multicasting in XTP as described in Revision 3.4 [PEI89].

XTP's multicast scheme provides low overhead and compatibility with unicasting by only having to set a bit in an outgoing data packet to indicate a multicast. Connection set-up works as with unicast, though a multi-endpoint connection is not as simple to manage as a single endpoint one. A potentially troublesome aspect arises in choosing the initial flow and rate parameters in the initial set-up packet, called a FIRST packet in XTP. Since these parameters apply to all receivers, they must be chosen to accommodate the weakest receiver, else costly go-back-N retransmissions will severely impede the forward progress of the delivery until the flow and rate parameters can be adjusted. Hence, to be safe, a multicast source may have to be initially very conservative.

Certain questions remain unanswered about the semantics for connection termination under multicast in XTP Revision 3.4. Under the current definition it would seem a single receiver could initiate and close a connection, possibly abruptly by setting the END bit in the trailer of an acknowledgement. A *graceful close* for a pair of contexts is a connection termination in which a receiver reliably receives all outstanding packets. In multicasting it is

desirable to have the ability to perform a graceful close between the sending context and all the receiving contexts. Yet, XTP provides no provision for a multicast graceful close. As depicted, the multicast source will simply respond to any single incoming CNTL packet with the correct termination bits set. One solution, since all CNTL packets are multicast, would be the addition of an ability in receivers to generate some response when they detect that some other receiver is initiating a termination of its connection.

Reliable multicast in XTP utilizes a go-back-N retransmission method. Receivers generate control packets upon receiving corrupted or out-of-sequence packets. Error control in the protocol uses multicast acknowledgements and hence assumes that a sender always belongs to the group to which it is sending. The receiver-generated control packets, sent as XTP CNTL packet types, contain the sequence number, called *rseq*, that is one plus the highest consecutive sequence number received without error. That is, *rseq* marks the beginning of the first gap in the data. Other receivers overhear the multicast control packets and dequeue any control packets that they have which contain an *rseq* value greater than or equal to the overheard value. This is called *damping*.

The receivers generate control packets for every out-of-sequence packet that arrives. This ensures that lost control packets are not a problem as long as the sender continues to transmit. In the last outgoing data packet of a message a bit is set that requests an explicit positive acknowledgement when that packet is delivered to the application process at the receiver. These NACKs, sent in CNTL packets, are subject to the same damping process as described above. The timer used whenever a unicast sender forces an explicit acknowledgement from its receiver is used to ensure repeated retransmissions of the acknowledgement request.

In order for damping to be effective, group members must experience nearly simultaneous delivery. This fact precludes the involvement of routers in an efficient multicast delivery

Otherwise, in scenario 2, if some receivers are far from their neighbors, these receivers will all escape the damping effect and multicast a control packet calling for the retransmission of packet 3 to be sent. This action could create a large number of multicast packets, both redundant CNTL packets and redundant retransmissions of packet 3, on the network at the same time. This wastes bandwidth and processing time at S. Moreover, in a case where out-of-sequence packets continue to stream in at the receivers, the failure of damping causes an explosion of redundant multicast packets.

The mechanism to determine when to release buffers at the multicast source in XTP seem likely to depend on estimates of the roundtrip time to the receivers. This roundtrip time must be determined from CNTL messages flowing back to the source over a single control channel shared by all members of the receiving set. Consequently, wide variance in the roundtrip times of group members due to members being spread across routers would force either very conservative buffer management or leaving open the possibility of releasing buffers before a retransmission request from a remote receiver arrives.

The worst case for scenario 3 would have R1 issue a CNTL packet that preempts CNTL packets from R2 and R3, but is then lost. Since no more data packets are arriving, the receivers will not reissue another CNTL packet upon receipt of the next data packet. At this point the sender takes over. The sender sets a timer whenever it issues an acknowledgement request (in XTP parlance, it sets the SREQ or DREQ bit, which it will have done in the trailer of packet 4). When this timer expires, a CNTL packet will be multicast containing the current time in its *sync* field; at each receiver, upon receipt of the CNTL packet, a response CNTL packet with an *echo* field containing the time in the incoming *sync* field is sent back to the sender indicating the gap in the received data at packet 3. These CNTL packets undergo the same damping process. Then the sender multicasts back packet 3 and the transfer is complete. The high overhead of control messages in this case comes from the fact that we have examined an end condition.

Usually the loss of a single data packet will be followed by a series of out-of-sequence deliveries at the receivers, and receivers generate a CNTL packet for every out-of-sequence data packet. Hence the resynchronization process carried out by *sync-echo* exchanges will be relatively rare.

XTP does not explicitly check for receiver failure. If R3 drops out during the multicast distribution, source S will not notice. The source is driven by control information received over a single response channel from the receivers as a whole and does not try to monitor the individual receivers at the multiple endpoints of the connection.

6.2. VMTP

The Versatile Message Transport Protocol (VMTP), recently revised [CHER89], is designed as a next generation protocol to accommodate communication strongly oriented toward request-response behaviour. Two examples of such communication environments are distributed on-line transaction processing systems, such as those used for credit card transactions, and clusters of workstations and file servers connected in a high speed LAN for page-level file access and Remote Procedure Call [CHER89]. VMTP's development has been coupled with the V Distributed System [CHER88], a distributed operating system, and together the two form a distributed system that provides a fully integrated multicast facility, i.e., not only a RMTP but an extensive user interface and group management scheme as well. The V System design heavily influences VMTP's support for reliable multicast.

A transaction starts with a client issuing a request to a server entity. Multicasting is simply a request that is sent to a group entity. At the server, upon receipt of the request, a transaction record is created. Ordinarily, a response packet containing the user-level response data is generated quickly at the server and functions as an acknowledgement to its associated

request. The transaction record is retained at the server for some time after transmission of the response for two reasons. First, if the client issues another request right away, this same transaction record can be used. Second, the transaction record is used to discard delayed duplicates of the request. Otherwise a new transaction record would be set up and the request processed again unnecessarily. Thus, on-demand connection set-up and the frequent use of user-level responses are two important features of the transaction paradigm.

In VMTP a receiver has no overhead associated with participating in a multicast as opposed to a unicast transmission; a receiver acts without knowledge of or regard for the actions of other members of the multicast group. Thus, unlike in XTP where receivers dequeue redundant error control messages upon overhearing other receivers' messages, receivers in VMTP are decoupled from each other. One implication is that VMTP supports a client not having to be a member of the group to which it is sending. Another aspect of not depending upon receivers communicating is that members may be dispersed widely, possibly across routers, subject to other constraints discussed below.

In a multicast, according to the semantics of the V System, a V System user unblocks upon receipt of the first response and thereafter may request further responses with a user-level primitive. The protocol continues to try to gather responses until another request to the same server(s) is initiated. The significance placed on getting the first response comes from the principle of allowing the user to control the degree of reliability completely and hence explicitly request responses beyond the first one. The V system does not support completely reliable multicast transmission as a primitive, and hence individual servers dropping out during an exchange will either be unimportant or handled by mechanisms above VMTP. Given these semantics, VMTP does not attempt to detect receiver failure.

In VMTP, the client focuses on obtaining the first response, due to the significance of the first response to higher layers and to preserve compatibility with unicasting. It sets its only timer upon issuing a request. Receipt of the first response disables the timer and thereafter responses are collected without any prodding from the client. If the timer expires before a response arrives, however, the client issues a demand for immediate acknowledgements from the servers. The client's demand and the servers' acknowledgements are sent as datagrams using ordinary request packets with a no-response bit set.

In scenario 1 then, S sends a request to the group entity containing R1, R2, and R3. If the user-level responses are not forthcoming before a timeout at S, then S sends a datagram request packet demanding immediate acknowledgement. (If this request fails to reach any server, the protocol falls back on higher layer error control to time out the entire request.) All three receivers respond. The client is assured that the requests have reached the servers and awaits responses. If at least one response is timely, the client will pass up the first response to the transport layer user and buffer the other responses as they arrive. Either case is expensive in terms of bandwidth. The first case should be rare since VMTP is intended for traffic that consists predominately of short, fast exchanges and multiple servers increase the chance of a quick response.

VMTP apparently depends upon the variance in remote processing times of a request to spread out responses and thus avoid overrunning the client. This strategy does not work with transport layer acknowledgements. While these should rarely be issued, they will be transmitted nearly simultaneously, and hence a large group on a single segment risks overrunning the source. Some user-level responses in any given transaction may be slow enough that they will not be ready to transmit until the transaction record for their request has been discarded due to the arrival of a new request from the same client. This represents a very limited form of damping.

VMTP is optimized for short transfers. The new version of the protocol incorporates a streaming mode in which a series of requests can be issued with asynchronous collection of responses. The intent is to allow, for example, file transfer as a series of requests with responses acting as acknowledgements. A multicast using the streaming mode would have to incorporate new mechanisms to count the responses from individual receivers since processing only a portion of the requests that make up the file transfer will be useless. In effect, the series of requests must be tied together into an atomic unit. Since these issues are not addressed in VMTP's multicast scheme, they must be handled at some higher layer.

In scenario 2, the unicasting of NACKs for packet 3 will again result in three separate requests for retransmission of the same packet. There exists no mechanism for suppressing the duplicate retransmissions. A packet lost by all receivers will thus result in a number of duplicate retransmissions as large as the multicast group size. Hence, while the loss of any single acknowledgement (e.g., scenario 3) will be covered by the other acknowledgements, the penalty for preserving simplicity in the receivers and the sender by making no attempt to coalesce acknowledgements is high in terms of bandwidth.

It is not made explicit in the most recent revision of VMTP as described in [CHER89] whether retransmissions are to be multicast or unicast during a multicast exchange. Unicasting seems acceptable except, as pointed out in scenario 2, a large number of receivers missing a packet would cause many duplicate packets to be framed and transmitted at the source. Multicasting then makes more sense, though again it would seem desirable to have some way at the source to coalesce requests for retransmission.

The VMTP protocol contains a couple of innovative features that seem intended specifically for multicast. It supports *co-resident addressing*, which allows a server that knows only the address of its client to address a response, e.g. for authentication, to the group manager

module on the client's host. Also, a user has the ability to ıfy that a multicast message should only be delivered at a particular server if the server is able to process the request immediately. Finally, a server can be instructed to dispose of the response as soon as the response is sent; retransmission of the request is handled by redoing the request instead of resending the original response. This option could clearly be useful in time management.

6.3. CP

In [CROW88] the authors present a RMTP, whic.: ve will call CP. They discuss the user level interface as well as the messaging protocol. The service interface of the protocol offers several types of services, ranging from the collection of a single reply from any one of a set of servers to the collection of all replies from all known servers. The send primitive reliably delivers the message to all receivers and the receive primitive allows the user to specify which and how many receivers must respond. If the client specifies that responses are needed from fewer than all possible receivers, then the protocol locks onto servers as they respond and, when the required subset is responding, informs all outstanding servers to discontinue working on the request.

The strategy of pruning the multicast group before all responses have been received is predicated on members of the group not being allowed to leave the group during an exchange. The responsibility for assuring group membership stability for the duration of each single multicast exchange is delegated to the group management scheme, whose details are not considered by the authors. Moreover, CP operates under the assumption that the group management scheme will report any receiver failures to the protocol. It should be noted that providing this functionality in the group management scheme is nontrivial.

The underlying network service is assumed to provide multi-destination addressing and routing capability, and the paper describing CP deals with the behaviour of the protocol over both LANs and LANs connected by WANs. The possibility of point-to-point connections in the network influences design decisions in CP. For example, allowing remote and local members in a group eliminates the option of multicasting acknowledgements for receiver interaction. Groups are thus open as in VMTP.

While groups spread over more than one segment are supported, distribution to such a group will be slow for the following reason. A retransmission timer at the multicast source is set to the upper bound of the roundtrip times estimated for each destination, but roundtrip times over an internet are much more difficult to estimate than within a single segment LAN. An incorrect setting will cause repeated retransmissions while a conservative estimate penalizes all receivers, especially local ones.

As in VMTP then, the receivers do not attempt to monitor each other's state, but unicast control packets that acknowledge data and carry the receiver's advertised window. The source employs a coupled multiple window strategy, wherein it keeps both an aggregate transmit window for the group and a separate transmit window for each of the servers. The overall window is the lower bound of the smallest of the advertised receive windows. Receivers attempt to piggyback acknowledgements onto the next message traveling back to the source with a timeout to trigger explicit acknowledgements if the delay is too long. Hence, the protocol accommodates request/response communication.

Retransmissions are driven by both the transmitter and the receivers. A receiver can ask for selective retransmissions up to a limit controlled by the source using the separate window for that receiver. The source unicasts back the missing packets and updates the receiver's window. When the retransmission timer expires at the source, the source calculates the

proportion of hosts not responding and either multicasts or unicasts retransmissions depending on whether the proportion exceeds a threshold. The threshold value is set depending on distance and the number of servers, being set lower for LANs and higher for internet usage, since broadcast LANs achieve multicast much more cheaply than internets.

Receivers unicast acknowledgements and set a timer to ensure that the acknowledgement is reliably delivered. Hence in scenario 2, each receiver would unicast a control packet for the retransmission of packet 3. At the source, the arrival of three requests would trigger a multicast retransmission. If the retransmission did not arrive before the expiration of the timer at a receiver, the receiver would re-issue its request for packet 3. Upon receipt of this acknowledgement, the source would decide whether to unicast or multicast packet 3 again. Coverage for lost acknowledgements is variable since, in scenario 3, the loss of any single acknowledgement would be insignificant if the source multicasts the retransmission of packet 3.

A multicast source in CP can lock onto receivers as they reply to a multicast request. Given information on exactly which receivers and how many receivers must respond, the protocol issues a low-level abort message to remote servers when all necessary servers have responded. This feature, which the authors propose as an experimental one, seems flawed in the following ways. First, a truly general voting function would probably involve passing code to the protocol, and distributing code is difficult. Second, in any protocol and especially a real-time one, data should be made available to the receiving application as soon as it arrives. Hence the application can handle all decisions about how many responses are enough, freeing the application from time-consuming interpretation, except that the application must know when to check for responses if it does not continually poll for them. Hence, a mechanism to notify the application when a good time to check the set of received responses would serve the same function as a low-level voting function and reduce the protocol's complexity with marginal loss of efficiency.

CP does struggle with the details of how it will provide for collection of replies, or, equivalently, how it will achieve many-to-one transmissions. Suggestions are made about the way in which an implementation might randomize replies to avoid overrunning the source, but, by the authors' own admission, difficulties arise. LANs demand strategies with extremely little computational overhead. Buffering at the source follows the same multiple coupled windowing strategy as with the send primitive (multicasting), providing synchronization by not letting any one receiver get too far ahead of the others. This in turn allows a host receiving many replies to pass parts of each reply, *in order* and also *in step*, up to the user.

6.4. NAPP

In [RAMA87] the authors start with the premise that in a LAN-based multicast the failure of one receiver to receive a packet strongly suggests that there are other receivers which have missed the packet. This tenet, along with the idea of making a multicast receiver completely responsible for ensuring that it reliably gets a sequence of messages, leads to the proposal of an RMTP based on negative acknowledgement with periodic polling, which we will refer to as NAPP.

NAPP operates under a complex set of rules. The source simply transmits data and performs retransmissions based on control information from the receivers. There are three packet types, ACK, PACK, and SREJ, for control information. 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 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. M will be the maximum number of packets that can be outstanding, that is, pending to be acknowledged at any one time. V_i will be the first in-sequence packet not received at receiver i. W_i is the window of receiver i consisting of V_i, ..., $V_i + M - 1$. 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 $V_i - M$, ..., $V_i - 1$. 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 $\in$ W_i, or upon receipt of a PACK(m), m $\geq$ 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(m) 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 $V_i - M, \ldots, V_i - 1$ 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 we have not specified NAPP completely and throughly, the above description gives the flavor and the most important aspects of its operation. In an actual implementation of NAPP, much attention will have to be given to the mechanisms to determine the correct settings for its many timers; the paper describing NAPP does not focus on these implementation details, but instead notes the relative lengths of timers, e.g., $T_{pack} > T_{ack}$. Since timers in the receivers are relative to the source, every multicast distribution will require some initialization process to set timers correctly. Short transfers, as with scenario 1, should not have to incur too much overhead in this regard since they will probably not need the full power of the error control mechanisms, e.g., ACK packets. On the other hand, very conservative or default initial settings may significantly degrade delivery service until settings are updated.

To illustrate the possibilities, consider scenario 2. Upon the arrival of packet 4, an SREJ(3) is issued at each receiver. Regardless of arrival times at the receivers, the SREJ packets will only be transmitted with probability P. If P is, say, 0.5, there is a good probability that one receiver at least, say R1, actually transmits its SREJ(3). Note that knowing the multicast group size would be very helpful in finding an optimum value for probability P; but then radical changes in group membership (e.g. a single site failure for groups with few members) would require a re-evaluation of the value of P as quickly as possible. The other two

Table 1
Summary of 4 RMTPs' Features

ATTRIBUTE	XTP	VMTP	CP	NAPP
GROUPS	closed	open	open	closed
UNICAST COMPATIBLE	low overhead	low overhead	not addressed	not addressed
PARADIGM	connection oriented	request/ response	connection oriented	connection oriented
ACKS	multicast	unicast	unicast	multicast
RETRANSMISSIONS	multicast	?	multicast/unicast	multicast
RECEIVER FAILURE (during an exchange)	not detected	not detected	group manager reports to protocol	not detected
ROUTING	no, ill-suited	yes, not ruled out	yes, explicitly considered	no, ill-suited

receivers, R2 and R3, will have scheduled their SREJ(3) packets for later transmission. Upon seeing the SREJ(3) from R1, each will reschedule the pending SREJ(3) and the pending PACKs at each site. The source will then receive the SREJ(3) packet, update the bottom of its transmit window to 2 since it can see that all receivers have received packets 1 and 2 from the state vector in the SREJ, and multicast packet 3 to the group. Note that should R1's SREJ(3) packet have been lost, i.e. scenario 3, that either a PACK or a SREJ at one of the three receivers would have been forthcoming within the minimum of T_{pack} and T_{srej} (assuming the first SREJ(3) is actually transmitted).

6.5. Summary

As shown in Table 1, the XTP multicast scheme limits itself to closed groups whose members are not separated by any routers. It seeks to attain maximum efficiency by taking full advantage of the broadcast medium and nearly simultaneous delivery to all receivers. Reliance on having a single control channel back to the sender from the receivers eliminates the possibility of monitoring the state of individual receivers. Thus, timing dependencies in the communication both between sender and receivers and between receivers must be used to handle key issues such as the release of transmit buffers and damping. The central question raised therefore is whether the advantage of having a single control channel is worth the price of abandoning multicasting through routers and detecting receiver failure.

NAPP also assumes closed groups and no routing. The protocol is even more sensitive to timing dependencies than the XTP scheme since the timers are intricately related. Such a scheme would adapt slowly to changes in either the multicast group or in the network environment. For short transfers the overhead of setting or negotiating the setting of timers may well be unacceptable. Long exchanges can benefit from the minimization of the amount of control traffic, but at the expense of a great deal of complexity in the receivers.

VMTP passes up a good deal of the burden for ensuring reliable multicast to higher layers, to the extent that VMTP itself retains little responsibility. The emphasis in VMTP on handling request delivery results in less attention to the kinds of optimizations found in protocols concerned mostly with stream transfers. The multicast scheme in VMTP reflects this in its use of datagram unicast acknowledgements and its lack of concern with retransmission policies.

CP achieves the most general purpose multicast scheme of the four investigated. Assuming the ability of the group manager to monitor group membership, CP offers a range of services and explicitly considers request/response behaviour, the possibility of point-to-point links in the delivery path, and efficient retransmission policies. What is not clear is how expensive some of the mechanisms, e.g., the multiple coupled window strategy, would be when integrated with a unicast protocol and whether the use of unicast acknowledgements could ever satisfactorily approximate the performance of a scheme like XTP's. In other words, the general nature of CP's multicast scheme trades off a great deal of performance in the important special case of closed groups on a single segment.

7. Conclusions and Future Work

In this paper we have identified the set of problems with which one-to-many communication facilities deal and looked at approaches and solutions in the literature. Any such facility consists of three parts: a group management scheme, a user level interface, and a RMTP. We conclude from our investigation that it is of vital importance that these three elements be carefully integrated in order to achieve efficient reliable multicast data transfer. A poor design for any one of the three components cannot be compensated for in the design of the other two.

An essential part of the discussion of reliable multicasting is a clear understanding of what *reliable* means in this context. Reliable transfer refers to delivery to some subset of the multicast group, an ideal receiver group. The user specifies to the best of its knowledge the ideal receiver group; the protocol attempts to ensure delivery to the set of destinations defined by the ideal receiver group, though in practice the multicast will affect some number of receivers outside of the ideal receiver group. The special case of an all-reliable transfer is especially difficult in that it forces the multicast source site to have the explicit address of all receivers. Moreover, group management semantics are needed to assure membership stability during an exchange and to detect host failures. Providing this functionality in the group management scheme is complex and costly in terms of message latency. As a result, RMTPs for real-time environments seem to have little hope of guaranteeing more than delivery to the set of sites belonging to the group at the beginning of the transfer and functional for the duration of the exchange.

Given the traditional connection-oriented communication paradigm, the success of any multicast scheme will depend heavily on being able to select groups that are homogeneous in their delivery characteristics. Inevitably, group delivery is dominated by the weakest member of the receiving set through flow and rate control parameters, settings for retransmission timers, transmission window management in the source, and other mechanisms that treat the multi-endpoint as a single entity. The solution to this problem is either to find another paradigm altogether or to partition the multicast into homogeneous receivers at a level above the transport layer. The latter avoids the difficult and expensive procedure of dealing with a weak receiver that is bogging down the entire multicast distribution dynamically during transmission.

Introducing routing complicates the multicast protocol considerably due to the high variance in roundtrip times to individual receivers and the loss of timing assumptions that allow damping and other useful forms of communication between receivers. An interesting question

that remains open is to what extent a multicast facility designer could take advantage of a restriction on the routing environment such that any packet experiences a maximum of N hops (where N is at most four) during delivery. We believe that this particular scenario for multicasting through routers merits further exploration.

Finally, we recognize that one-to-many communication, while an important (and still problematic) starting point, does not represent the full extent of the N-party services that will be demanded in future communication protocols. The problem of a many-to-one transfer, or *de-multicasting*, has not been adequately explored. We intend to investigate how de-multicasting and multicasting might be made the basis of a protocol in which the underlying paradigm is not connection-oriented. We believe that additional work on communication paradigms will shed new light on multicast and de-multicast operations.

REFERENCES

AHAM88 Ahamad, M., Ammar, M.H., Bernabeu-Arban, J.M., and Khalidi, M., "Using Multicast Communication to Locate Resources in LAN-Based Distributed System", *Proceedings of the 13th Conference on Local Computer Networks*, IEEE, Minneapolis, Minnesota, 1988.

AGUI84 Aguilar, L., "Datagram Routing for Internet Multicasting", *ACM Computer Communications Review*, Volume 14, Number 2, pp. 58-63, 1984.

AGUI86 Aguilar, L., Garcia-Luna-Aceves, J., Moran, D., Craighill, E., and Brungardt, R., "Architecture for a Multimedia Teleconferencing System", *ACM 1986 Symposium on Communications Architectures and Protocols*, Stowe, Vermont, pp. 126-135, August 1986.

BACK88 Backes, Floyd, "Transparent Bridges for Interconnection of IEEE 802 LANs", *IEEE Network*, Volume 2, Number 1, January 1988.

BERG85 Berglund, E.J., and Cheriton, D., "Amaze: A Multiplayer Computer Game", *IEEE Software*, Volume 2, Number 3, May 1985.

BIRM87 Birman, Kenneth, and Joseph, T., "Reliable Communication in the Presence of Failures", *ACM Transactions on Computer Systems*, Volume 5, Number 1, pp. 47-76, February 1987.

CHAN84 Chang, Jo-Mei and Maxemchuk, N., "Reliable Broadcast Protocols", *ACM Transactions on Computing Systems*, Volume 2, Number 3, pp. 251-73, August 1984.

CHER85a Cheriton, D. and Deering S.E., "Host Groups: a Multicast Extension for Datagram Internetworks", *Proceedings of the Ninth Data Communications Symposium*, IEEE/ACM, Whistler Mountain, BC, Canada, pp. 172-179, September 1985.

CHER85b Cheriton, D. and Zwaenepoel, W., "Distributed Process Groups in the V Kernel", *ACM Transactions on Computer Systems*, Volume 3, Number 2, pp. 77-107, May 1985.

CHER89 Cheriton, D. and Williamson, Carey L., "VMTP as the Transport Layer for

High-Performance Distributed Systems", *IEEE Communications Magazine*, pp. 37-44, June 1989.

COOP84 Cooper, E.C., "Circus: A Replicated Procedure Call Facility", Fourth Symposium on Reliability in Distributed Software and Database Systems, 1984.

CROW88 Crowcroft, J. and Paliwoda, K., "A Multicast Transport Protocol", *Computer Communication Review*, Volume 18, Number 4, pp. 247-256, August 1988.

DALA78 Dalal, Y. and Metcalfe, R., "Reverse Path Forwarding of Broadcast Packets", *Communications of the ACM*, Volume 21, Number 12, pp. 1040-1048, December 1978.

DEER88 Deering, S.E. and Cheriton, D., "Multicast Routing in Datagram Internetworks and Extended LANs", *Computer Communication Review*, Volume 18, Number 4, August 1988.

DEUN83 DEUNA User's Guide, "EK-DEUNA-UG-001", Digital Equipment Corporation, 1983.

ETHE82 Ethernet: A Local Area Network — Data Link Layer and Physical Layer Specifications, Digital Equipment Corporation, Intel Corporation, and Xerox Corporation, Version 2.0, November 1982.

FRAN85 Frank, A., Wittie, L., and Bernstein, A., "Multicast Communication on Network Computers", *IEEE Software*, pp. 49-61, May 1985.

GAIT89 Gait, Jason, "A Kernel for High-Performance Multicast Communications", *IEEE Transactions on Computers*, Volume 38, Number 2, February 1989.

HUGH87 Hughes, L., "An Introduction to Multicast Communication", *Proceedings of the Special Conference of the International Council for Computer Communication*, New Delhi, India, October 1987.

IEEE85 Institute for Electrical and Electronic Engineers, Project 802— Local and Metropolitan Area Network Standards, "IEEE Standard 802.1 (D) MAC Bridges", 1985.

ISO7498 International Organization for Standardization, "Information Processing Systems — Open Systems Interconnection — Basic Reference Model", *Draft International Standard 7498*, October 1984.

MICR86 **Microcommunications Handbook**, Intel Corporation, ISBN 1-555-12-007-7, 1986.

PALI88 Paliwoda, K., "Transactions Involving Multicast", *Computer Communication Review*, Volume 11, Number 6, pp. 313-18, December 1988.

PEI89 Protocol Engines, Incorporated, "XTP Protocol Definition Revision 3.4, Santa Barbara, California, July 1989.

RAMA87 Ramakrishnan, S. and Jain, B., "A Negative Acknowledgement with Periodic Polling Protocol for Multicast over LANs", *IEEE INFOCOM 1987: The Conference on Computer Communications Proceedings*, IEEE, San Francisco, California, April 1987.

SCHM88 Schmuck, F., "The Use of Efficient Broadcast Protocols in Asynchronous Distributed Systems" (excerpts from), Ph.D. Thesis, TR 88-928, Cornell University, Ithaca, New York, August 1988.

SINC88 Sincoskie, W.D., and Cotton, C.J., "Extended Bridge Algorithms for Large Networks", *IEEE Network*, Volume 2, Number 1, pp. 16-24, January 1988.

STAL87 Stallings, W., **Handbook of Computer Communications Standards, Volume 2: Local Networks Standards**, Macmillan, New York, 1987.

WALL80 Wall, D.W., "Mechanisms for Broadcast and Selective Broadcast" (excerpts from), Technical Report 190, Computer Systems Lab., Stanford University, June 1980.