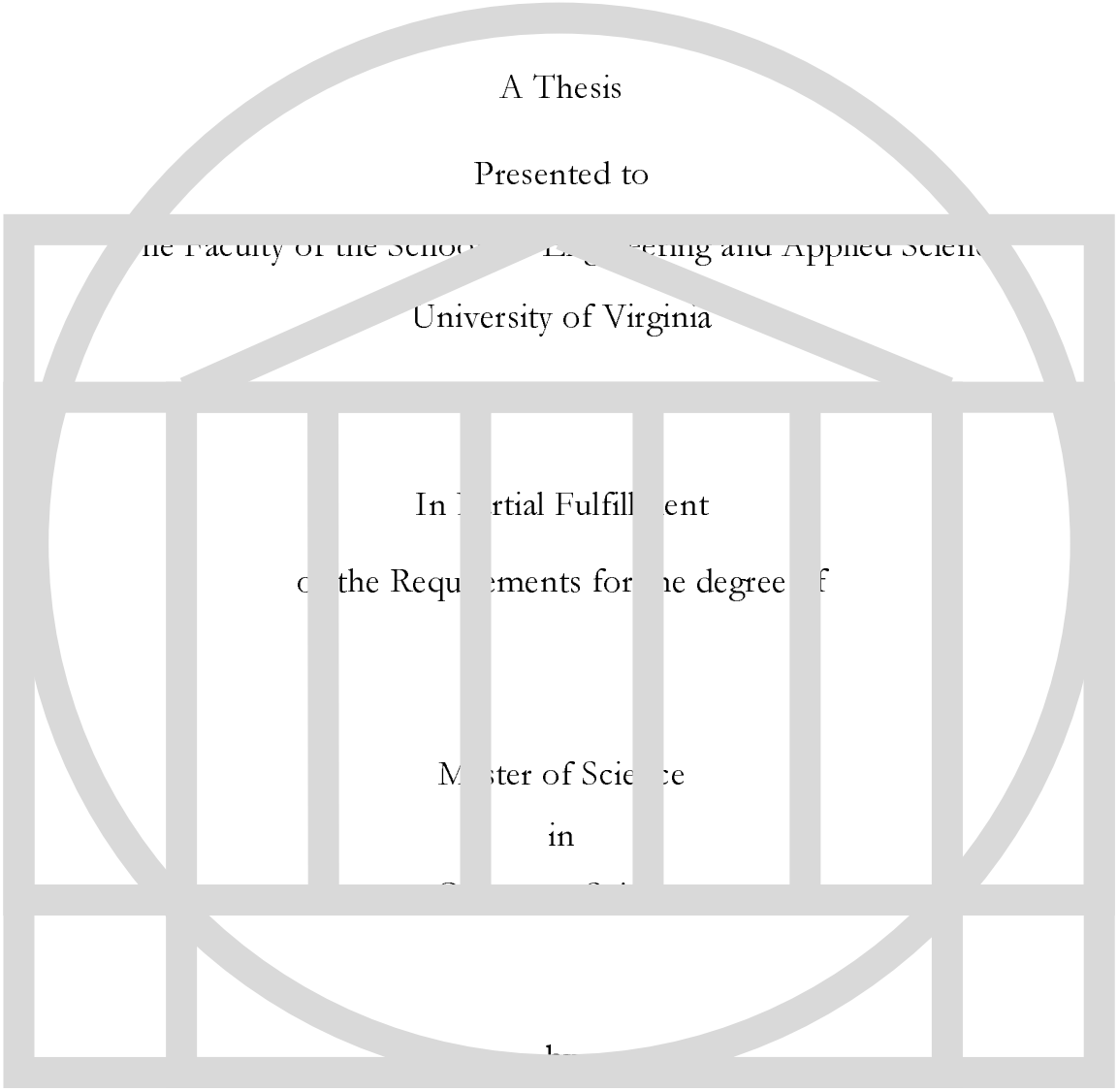
Reliable Multicast Services For Tele-collaboration



A Thesis

Presented to

the Faculty of the School of Engineering and Applied Science

University of Virginia

In Partial Fulfillment

of the Requirements for the degree of

Master of Science

in

Computer Science

by

David G. Bassett

January 1997

Approval Sheet

This thesis is submitted in partial fulfillment of the
requirements for the degree of
Master of Science (Computer Science)

David G. Bassett

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

Jörg Liebeherr, Advisor

Alfred C. Weaver, Chairman

James P. Cohoon

Accepted for the School of Engineering and Applied Science:

Dean, School of Engineering and Applied Science

January 1997

Abstract

This thesis presents a new approach to reliable multicasting tailored to meet the requirements of tele-collaborative applications. We explore the issues of reliable multicast through a systematic analysis of the requirements of tele-collaborative applications. As a result of this analysis, we present a reliable multicast design space consisting of the following dimensions: ordering, reliability, group size, group membership, persistence, and receiver storage. Persistence, a dimension referring to the lifetime of packets at senders in receiver-initiated reliable multicast protocols, is introduced as a new dimension. We show that persistence overcomes many of the drawbacks of previous receiver-initiated protocols. We present a novel approach referred to as Logical Persistence in which packets are stored based on a logical partitioning of data. We also present a new protocol called the Tunable Multicast Protocol (TMP). This protocol provides a wide collection of services that can be specified by a number of tuning parameters. TMP provides a reusable and flexible reliable multicast protocol capable of efficiently supporting a wide variety of tele-collaborative applications.

TABLE OF CONTENTS

Abstract iii

Acknowledgements viii

1. Introduction 1

2. Background 7

2.1. A Taxonomy of Reliable Multicasting Protocols 7

2.1.1. Sender-Initiated Protocols 9

2.1.2. Receiver-Initiated Protocols 18

2.2. Internet Multicast Backbone (MBone) 22

2.3. Real- Time Transport Protocol (RTP) 28

3. A New Approach to Reliable Multicasting 31

3.1. A Reliable Multicast Design Space 33

3.1.1. Ordering 35

3.1.2. Group Size 37

3.1.3. Group Membership 39

3.1.4. Reliability 41

3.1.5. Persistence 43

3.1.6. Receiver Storage 47

3.2. Definition of Existing Protocols within the Design Space 49

4. The Tunable Multicast Protocol 52

4.1. TMP Ordering Services 56

4.2. Error Correction 57

4.2.1. NAK Suppression 59

4.2.2. Response Implosion Avoidance 61

4.2.3. Combining RTP with a Reliability Mechanism 63

4.3. Rate Control 66

4.4. Persistence 68

4.5. The TMP Application Program Interface (API) 71

4.6. Internal MSocket Object Model 79

4.6.1. MSocket Object 80

4.6.2. MStream Object 83

4.6.3. MWriter Object 83

5. Conclusions 85

5.1. Summary of Contributions 86

5.2. Future Work 87

Bibliography 88

Appendix A - TMP Packet Format A-1

LIST OF FIGURES

Figure 1 - Internet Multicast Backbone 23

Figure 2 - A Reliable Multicast Design Space Model 33

Figure 3 - RMP in the Reliable Multicast Design Space Model 49

Figure 4 - SRM in the Reliable Multicast Design Space 51

Figure 5 - Definition of TMP in the Design Space Model 52

Figure 6 - Error Correction Thread Algorithm 59

Figure 7 - Repair Response Thread Algorithm (Receiver Storage Mode) 61

Figure 8 - (n,m) Persistence Windows 68

Figure 9 - MSocket Object Interface 73

Figure 10 - MSocket Object Diagram - Major Components 79

Figure 11 - MStream and MWriter Objects 82

Figure 12 - Probabilistic Error Correction Thread Algorithm 89

LIST OF TABLES

Table 1 - IETF TTL Conventions 24

Table 2 - Reserved Application Type Values 78

ACKNOWLEDGMENTS

I would like to thank my wife, Jolanthe, for her encouragement, understanding, and patience. Also, an immeasurable amount of appreciation and gratitude goes to Jörg Liebeherr for his unbelievable support and patience. Without his steady assistance, guidance, and encouragement, this thesis would not have been written. I would like to thank Lance Hopenwasser, Matt Lucas, and Steve Brown for their assistance in the original design, concepts, and implementation of the Tunable Multicast Protocol. Finally, I would like to thank the United States Army for giving me the opportunity to pursue my studies here at the University of Virginia.

1. Introduction

Modern computer networks are capable of providing very high bandwidth currently exceeding 100 megabits per second in both local and wide area networks. This tremendous bandwidth combined with the continuing increase in the power of personal computers and workstations has led to the increased popularity of networked-based multimedia applications. These applications transmit audio and video data as well as other forms of time-independent data such as text and images and allow users to work together, or collaborate, remotely using existing computer networks. We use the term tele-collaboration to refer to use of computer networks that enable groups of people to interact and work together remotely. Tele-collaboration tools now include audiocast [JAC92][RAJ92], videocast [DEM94], group video conferencing [FRE][TUR93], and distributed whiteboards [FLO95].

Current network protocols are not designed to support the high demands of tele-collaboration applications. The vast majority of data transmitted on the Internet today is *Unicast*, where data is transmitted between two stations only. Although it is possible to use unicast to support very small tele-collaborative groups, it is inefficient for larger group sizes. If unicast transmission is used for group applications, data must be transmitted to each member of the group individually, consuming valuable processing and network resources. Unicast transmission of data has been shown to be incompatible with the bandwidth demands of multimedia tele-collaboration applications for groups approaching as few as 10 members [KAA89].

Multicast Transport Protocols were introduced in the late 1980's to enable applications to communicate with many destinations at the same time. In multicast communication, a single station transmits a single copy of data to a defined group of receivers, or *multicast group*. By allowing a single transmission to be received by many stations, multicast protocols are able to more efficiently support the demands of high bandwidth audio and video applications. The *Internet Multicast Backbone* (*MBone*), an experimental multicast virtual running over the Internet, was established in 1992 [CAS94]. Today, the MBone consists of more than 2500 subnets, and 12000 routine users [DIO96]. By using multicast communications over the MBone, it is now possible for thousands of stations to receive live video and audio data from the same sender while conserving available bandwidth for other applications.

The Internet Multicast Protocol used by the MBone supports only an unreliable "best effort" data transport service. Packet loss is frequent on the MBone and has been measured at nearly 25% [ZAB96]. Since acceptable playback of video and audio can be accomplished despite the loss of some data, this loss rate has not been a significant problem for multicast audio and video applications. Other applications require that data packets be delivered reliably, i.e., in order, without errors and without duplication. Consider the case of a binary file that is multicast to all members of a multicast group. If any part of the file is lost, the entire file is unusable. A reasonable approach to introducing reliability into multicast communication would be to use well established unicast reliability services such as TCP [POS81]. However, if a sender is communicating with hundreds of receivers, it is clear that unicast protocols are inappropriate and inefficient if used for providing a reliable multicast service.

The problem of reliable multicast communications has been widely researched [CHA84] [BIR87] [STR92] [FLO95]. Despite the large body of research on reliable multicast protocols, the computer networking community has not converged to a single reliable multicast protocol capable of supporting the needs of all multicast applications. A primary reason that no consensus has been possible is that applications vary dramatically in their communications requirements. In this thesis, we conduct a systematic analysis of the requirements that tele-collaborative applications place on reliable multicast protocols. As a result of this analysis, we identify six primary dimensions that make up the *Reliable Multicast Design Space*. These dimensions include *Ordering, Reliability, Group Size, Group Membership, Persistence,* and *Receiver Storage*. There has been significant discussion of the first four dimensions in previous work [BIR87] [PIN94] [BOR94] [CHT96]. We introduce the two new dimensions of *Persistence* and *Receiver Storage* that apply to all receiver-initiated multicast protocols (i.e., those protocols in which each receiver is responsible for initiating the communications needed to ensure reliability). Data must be stored for some period of time so that it can be retransmitted to correct errors. *Persistence* refers to the lifetime of data kept for this purpose. The dimension of *Receiver Storage* refers to the amount of data that all receivers store so that error correction can be distributed to group members. By having all receivers store data, any station that has received a transmission successfully can retransmit the data to correct errors. Although other protocols have introduced this concept, we propose that a range of receiver storage be allowed rather than specifying that all transmissions must be stored indefinitely [FLO95].

We discuss three modes of *Persistence* and introduce a novel approach called *Logical Persistence*. Logical Persistence allows the application to divide data into logical partitions (e.g., each file in a multicast file distribution application or slides in a presentation). Each

sender stores a specified number of logical partitions for error correction and to enable new members to receive a "history" of data to bring them up to the current state. This method of persistence allows the application to store only that portion of the session history that is actually important rather than storing information for a specified period of time or some "window" of sequence numbers.

In this thesis, we propose a new approach to multicasting that is tailored to the needs of tele-collaborative applications. Early research into reliable multicast protocols was based on the needs and restrictions of distributed computing [CHA84][BIR87]. The services provided by many existing reliable multicast protocols place a large emphasis on establishing a total ordering of data and achieving perfect reliability. We do not believe that these services are appropriate for most tele-collaborative applications. Rather, we believe that total ordering services hinder the ability of existing protocols to support large groups efficiently [PIN94]. In our approach to reliable multicasting, total ordering is abandoned in favor of less stringent ordering services that are more easily and efficiently provided. Similarly, because absolute group membership guarantees are difficult if not impossible to achieve[CHT96], we propose that only a "best effort" reporting of group membership be provided. These services, although insufficient for distributed processing, are adequate for most tele-collaborative applications.

Based on our exploration of the multicast design space, we introduce a new protocol called the Tunable Multicast Protocol (TMP). TMP is based on a recent proposal to achieve reliable multicasting presented in [FLO95]. We use the packet format from the Real-Time Transport Protocol [SCH96]. TMP is designed to complement the software engineering process and speed the development of a wide variety of tele-collaborative

applications. We reject the popular belief that a single protocol is incapable of supporting a wide variety of applications [CLA90] [FLO95]. To overcome the inefficiencies inherent in a protocol providing a single service model, TMP uses a set of *tuning parameters* enabling applications to specify a wide range of services. We believe that through the use of tuning parameters, efficiency can be maintained while isolating application developers from the complexity of the reliability protocol. By providing a reusable C++ class implementation of the underlying reliability engine, the application programmer is shielded from the complexity of ensuring ordered delivery of data while, at the same time, reaping its benefits.

TMP's features evolved directly from our systematic analysis of the reliable multicast design space. TMP's tuning parameters provide flexibility in a number of important dimensions. Total ordering amongst members of a multicast group is sacrificed in favor of simpler, but more efficient, ordering services that guarantee in order packet delivery relative to each sender. Two flexible models of persistence are supported and discussed in detail in Section 4.4. Both models presented are specified through tuning parameters defining the amount of data stored at each sender as well as the window of data that receivers will attempt to receive reliably. These values can uniquely specify the persistence services for a wide variety of applications.

This thesis explores issues involved in reliable multicast communication and presents a new protocol specifically designed to meet the needs of tele-collaborative applications. In Chapter 2, we present a background and literature review of reliable multicasting protocols and multicasting on the Internet. In Chapter 3, we present our new approach to reliable multicasting. We explore each dimension of the reliable multicast design space and introduce the two new dimensions of *Persistence* and *Receiver Storage*. In

Chapter 4, we present our Tunable Multicast Protocol. We evaluate TMP in the reliable multicast design space (introduced in Chapter 3) and detail the range of tuning parameters available to the application developer through the TMP application program interface (API). We also provide an overview of the underlying C++ object structure that was implemented in the TMP protocol. Chapter 5 presents a summary of our results as well as directions for future work.

2. Background

Since the early 1980's, a wide variety of protocols has been introduced that address the goal of reliable multicast communication. In this Chapter, we will present a review of the literature addressing this prolific research area. Since the TMP protocol is designed to be implemented with the existing network support for multicasting, we will also introduce the Internet Multicast Protocol (Multicast IP). Multicast IP is implemented as an experimental virtual network known as the Multicast Backbone (MBone). We will discuss the routing and group membership protocols that are in use on the MBone today and discuss the future of this technology. Finally, we will discuss the Real-Time Transport Protocol (RTP) since its format and control protocol serve as a building block for the TMP protocol introduced in this thesis.

2.1. A Taxonomy of Reliable Multicasting Protocols

Today there exists a large number of protocols that provide a reliable transport service. These protocols differ both in their fundamental view of reliability as well as in their approach to implement a reliable service. For simplicity in the discussion of the protocols we will use the terms "sender" and "receiver" to refer to stations in a multicast group. A station may be both a sender and receiver within the same multicast group. Unless otherwise noted, all protocols apply to a "$n \times n$" multicast group in which all stations are allowed to send to all other stations in the group. Without multicast support at the

network and transport layers, the simplest way to accomplish reliable group communication is through multiple TCP unicast connections [POS81]. Although TCP provides ordered delivery from each sender to each receiver in the group, such a scheme is very inefficient because a packet must be transmitted once for each receiver in the group. By multicasting the data once instead of unicasting it multiple times, there is an obvious improvement in efficiency. Another critical issue to consider for evaluating any reliable multicast protocol is the algorithms that accomplish error control and correction. Specifically, a protocol must decide who is responsible for detecting errors and how to communicate that information within the group.

The two most frequent approaches to adding reliability to multicast protocols are commonly known as *sender-initiated* and *receiver-initiated* [PIN94][LEV96]. The primary difference between these two approaches is that in sender-initiated protocols, the sender keeps explicit track of the receiver set and verifies successful receipt of data. In receiver-initiated protocols, the burden of ensuring reliability is placed on each receiver. Receivers are responsible for requesting retransmission of missing or corrupted data. Senders do not verify successful transmission of data but rather assume this implicitly from the absence of retransmission requests. We will review a number of sender and receiver-initiated multicast protocols and discuss their drawbacks and advantages. The SRM protocol will be discussed in more detail as it forms the foundation for the TMP protocol.

2.1.1. Sender-Initiated Protocols

In sender-initiated protocols, the sender of the data is responsible for ensuring that each receiver in the multicast group receives the data reliably. There are a number of required tasks to ensure reliability. Since the sender must ensure delivery, the sender must have explicit knowledge of the multicast group membership. Also, the receivers must have some means of communicating to the sender that packets have been successfully received. If receivers individually acknowledge each packet (or alternatively groups of packets) by sending an acknowledgment packet (ACK) back to the sender, the sender can quickly become overwhelmed with the multiple ACKs generated by a large receiver set. This problem, known as the *Ack Implosion Problem* [DAN89] [JON91], prevents protocols that require explicit acknowledgement of every packet from supporting large group sizes[PIN94]. Sender-initiated protocols have addressed this problem through the establishment of tree or token-ring type structures for the transmission of data and ACKs. These protocols can be roughly subdivided into those that use a central "controlling" station to establish ordering and reliability and those protocols that use a distributed ordering algorithm.

The underlying theory behind most distributed sender-initiated protocols comes from the field of distributed computing. In particular, Birman's early work on the ISIS system was directed not towards an Internet communications architecture but to a "loosely coupled distributed environment"[BIR87]. Birman's ordering algorithm was based on a system of vector clocks first introduced by Lamport [LAM78]. Lamport defined the notion of a *causal* relationship between events in a distributed system [LAM78] and the "happened before" relation ($\rightarrow$) to capture causal dependencies between events. Birman used an extension of Lamport's vector clocks [LAM78] to develop the ISIS toolkit which

included a causal delivery protocol called CBCAST [BIR91] that established a total and causal ordering of messages in a distributed system. CBCAST provides totally ordered delivery by ensuring that messages are only delivered to a receiver if all prior messages have already been delivered. Messages are marked with a vector clock timestamp that establishes a causal ordering of all messages in the system. If a station has not received an acknowledgment for every packet that causally precedes the current packet, it will "piggyback" all causally dependent messages together with the new message. Although these "piggybacked" messages allow receiving stations to immediately process CBCAST transmissions, "piggybacked" messages may result in duplicate messages being transmitted unnecessarily. This redundant transmission of messages wastes valuable network and processing resources and seriously reduces the efficiency of the protocol. Furthermore, CBCAST requires ACKs for each transmission and is subject to the Ack Implosion Problem.

Another protocol that used a distributed approach to establishing reliable and ordered delivery was the *Psync* protocol [PET89]. *Psync* is not a reliable multicast protocol but rather could best be described as an ordering protocol. Psy*n*c uses an ordering algorithm based on the use of Context Graphs (CG) to provide total and causal ordering. CGs are directed acyclic graphs constructed such that the messages are represented by vertices and the causal dependencies of a given message with other messages in the system represented by the edges of the graph. Sending a new message involves adding a new message node to the graph. In order to add a node, the causal dependencies must be calculated. This is accomplished by taking the transitive reduction of the previous CG with the new node and dependency arcs [LUC95]. Messages are sent along with the information that specifies the message with the other nodes in the senders' CG. Using this edge

information, the receiving station can determine causal ordering. By sending only the information regarding the ordering relationships for a message rather than actual copies of prior messages, *Psync* obtains an efficiency advantage over ISIS. *Psync* requires each receiver to acknowledge every message and can quickly lead to the Ack Implosion Problem for even relatively small size groups. However, *Psync's* biggest drawback is that it fails to address the critical issues of fault-tolerance and host failure. There are no explicit procedures in the protocol to handle lost messages or lost acknowledgments. The failure (when detected) is merely forwarded to the application with no additional action taken by the protocol to ensure delivery. In addition to these problems with fault-tolerance, like ISIS Psync requires ACKs for each transmission and is subject to Ack Implosion for large group sizes.

The pioneering work on sender-initiated, totally ordered multicast protocols was presented by Chang and Maxemchuck [CHA84]. In the Chang and Maxemchuck Algorithm (CM), reliability and ordering are obtained through the use of a token ring technique. This protocol operates by passing a token around a ring formed by the members of the multicast group. This logical ring merely specifies the sequence that is followed as the token is passed between the members of the group as opposed to token ring networks in which the ring reflects the physical topology. The token identifies the member that is currently performing the duties of what might be thought of as a controlling site. Any member of the group can transmit a multicast packet, but only the station with the token transmits an ACK. When the token site receives a packet, it assigns the packet a global sequence number and multicasts the message to the entire group. Participants not holding the token are able to detect the loss of sequenced packets through the global sequence number and request retransmission from the token holding site. A station that is holding the token may not pass it to the next station on the logical ring until it has received all

messages up until the most recent sequence number. Once the token has been passed to all members of the group, positive acknowledgment of the previous round of packets can be inferred (otherwise the token could not have been passed). CM includes procedures for recovery in case the token holder fails. Although load distribution is good since there is no central site that must process all sequencing and recovery, there still exist significant drawbacks to CM when considering large groups. CM only discovers the failure of a non-token holding site once the token has been passed to every member of the group. Therefore, in the worst case, for a group of stations of size *N*, it may take *N-1* token transfers to detect a host failure. This is clearly undesirable if the number of stations, *N*, is large. Moreover, if the token holding site fails, recovery is extremely costly since all members of the group must agree on a new token holder. Finally, although CM appears at first glance to provide a complete causal ordering of messages, causal ordering is not guaranteed.

A number of protocols were developed that used a central controlling station to provide ordering and reliability. A good example of this method can be found in the Multicast Transport Protocol (MTP) [FRMA90]. In MTP, all stations request and receive a token from a central "Communications Master" station prior to sending. The Master site uses these tokens to both provide ordering and flow control. Stations only notify the Master when packets fail to arrive successfully. This negative acknowledgment (NAK) is a commonly used method to avoid ACK implosion. When any station requests retranmission of a missing transmission, the retransmission is multicast to the entire group. To correct errors, the Master must store packets so that they can be retransmitted when requested by group members. However, the Master station only stores packets for a finite amount of time and total reliability is not guaranteed. Once this time has expired, the Master station

will inform the receiver that the packet is unavailable. By depending upon a single central station, MTP is highly susceptible to the failure of the "Communications Master". Moreover, scalability is limited due to the communications bottleneck into and out of the controller.

MTP was one of the first protocols to address persistence and implemented temporal persistence by introducing three parameters [FRE90]:

- **Heartbeat**: Specifies a base unit of time in milliseconds.
- **Window**: The maximum number of data packets a sender can send during any heartbeat.
- **Retention**: The maximum number of heartbeats that a sender must store packets for possible retransmissions.

Note that the decision on when to discard data is defined strictly by the time the data was sent. Data is guaranteed to be available for exactly *Heartbeat*Retention* milliseconds. The first two parameters, *Heartbeat* and *Window*, specify rate control parameters (on a packet basis without regard to packet size). All sending stations must retain at most *Retention*Window* packets (in the case that a station is sending at its maximum transmission rate) but could store fewer packets than that in the case of stations sending at transmission rates lower than their maximum. Note that packets are stored only to enable error recovery. In Section 3.1.5, we will discuss the role of persistence not only in correcting errors but also so that a "history" of packets can be transmitted to new group members.

MTP-2 [BOR94] was introduced to address some of the drawbacks with the original MTP protocol. Some of the important features added by MTP-2 include:

- Advanced Joining Procedure: MTP's joining procedures specifies that new members are only admitted when the Master passes sending control from one sender to another. MTP-2 allows members to join at any time by ignoring packets with a sequence number transmitted before the new member joined.
- Tolerance to Master Failure: MTP has no procedures to recover from the failure of the Master station. MTP-2 adds a recovery procedure so that existing group members can agree on a new Master station and continue the session.
- Master Migration: In addition to providing procedures to recover from the failure of the Master station, MTP-2 adds procedures so that a Master can voluntarily give up its duties if it becomes overloaded or decides to leave the session. The duties of the Master are then transferred to an alternate station agreed upon by the group.

Although MTP-2 addressed the issues involving a single point of failure, a number of disadvantages remain. Since only a single station can have the token at a time, all other members are effectively blocked from transmitting. In the case that all members of the group have requested permission to transmit, the Master will sequentially grant permission to each member of the group. In a large group with N members, a station may have to wait for $N-1$ other members to transmit before receiving a token to transmit.

Another totally ordered, atomic multicast protocol introduced by Kaashoek [KAA89] is known as the Reliable Broadcast Protocol (RBP). RBP is similar to MTP in that it uses a central site, known in RBP as a *sequencer node*, to achieve ordering and reliability. All stations first unicast the message to the sequencer node. The sequencer node orders the message with a sequence number and multicasts the sequenced message to the group. Since all messages are marked with a sequence number, it is trivial for receiving stations to detect losses. Receivers unicast NAK packets to the sequencer node to request retransmission of missing packets. Since the sequencer node must buffer all packets transmitted so that errors

can be corrected, receiving stations periodically send a message that contains the highest sequence number they have successfully received. When all stations have received a packet, the sequencer discards the message, preventing the need for an infinite storage space. Unlike MTP, RBP allows all stations within the group to send without waiting for a token. However, as in the original MTP protocol, RBP will fail in response to the failure of the sequencer node and suffers from the same scalability problems as MTP.

The Xpress Transport Protocol (XTPv4) [STR92] [WEA95] provides perhaps the most flexible service primitives of all reliable multicast protocols. In XTP, a central agent known as the *Multicast Group Manager (MGM)* maintains a list of group members as well as a list of specifications regarding the error, flow and congestion control algorithms. XTP allows different levels of service depending on the needs of the application by separating the policies for the services provided from the protocol itself. As an example [WEA95], XTP allows each application to determine group membership policies by merely transmitting group membership information to the application rather than implementing a predefined and possibly inappropriate policy. Like RBP, stations first unicast messages to the MGM which in turn multicasts them to the group. XTP supports three types of error control:

1. Fully reliable, requiring acknowledgment of every packet.
2. Unreliable UDP-like service.
3. A special mode referred to as *fast negative acknowledgment* (*fastnak*) in which receivers immediately transmit a NAK in response to the detection of missing packets.

Each of these error control mechanisms has unique properties that can lead to problems if wrongly applied. In the case of fully reliable error control, XTP was one of the

first protocols to introduce slotting and damping to avoid ACK Implosion[1]. XTP "slots" acknowledgments by having each receiver delay for a randomly chosen interval before issuing an ACK. This prevents all receivers from responding in unison. Also, each receiver will "damp" (cancel) its control messages to other members if it determines that the content of its message is fully contained in the messages transmitted by other stations. The *fastnak* error control service was designed primarily for use on a LAN where out of sequence packets would normally be associated with a network loss rather than network delay. If *fastnak* is used in a typical large scale Internet environment, implosions of NAKs could result due to the immediate responses by receivers to a missing packet.

More recently, the Reliable Multicast Protocol (RMP) [MON94] was introduced based on the CM Algorithm. RMP overcomes some of the drawbacks with CM and provides a more flexible service. RMP uses the basic CM algorithm for its basic delivery of packets but provides a number of additional services including: support for multiple multicast groups as opposed to a single broadcast group, an implicit naming service that maps textual group names into communications groups, a hierarchy of reliability services including the totally ordered service provided by CM as well as a K-resilient[2] packet delivery in which reliable delivery is guaranteed to a subset of the group, improved group membership and host failure recovery algorithms, and window flow control and congestion control mechanisms based on Van Jacobsen's slow start algorithm[WHE94]. RMP has been successfully implemented on a variety of platforms including Unix and Windows95.

---

[1] Slotting and damping are introduced as suggested multicast heuristics and are not actually part of the XTP Protocol [STR92].

[2] In a K-Resilient service, delivery is guaranteed to at least *K* members of the group. This quality of service can be generalized to fully reliable if *K* is chosen equal to the group size.

Garcia-Molina and Spauster [GAR91] introduced a tree-based structure as an alternative to a logical token ring with the Message Passing (MP) Protocol. MP uses a propagation graph algorithm for multicast delivery. Messages are ordered as they traverse through the graph such that total ordering amongst members of the group is evident to the receiving stations. This algorithm is a clear departure from the previously discussed protocols in that there is no reliance on a Master site (or token holding site) to obtain a total ordering. Because ACKs are only transmitted to a node's parent rather than to the entire group, ACK Implosion is clearly avoided. However, the difficulty with this protocol as well as all other tree-based protocols is the overhead in constructing and maintaining the tree. Note that the tree must be dynamically modified both in response to host failures and to members joining and leaving the group.

In keeping with the tree-based structure of MP, Yavatkar [YAV95] presented the Tree-based Multicast Transport Protocol (TMTP). This protocol addresses the case of a single-sender ($1 \times N$) session referred to as a *dissemination group*. Prior to transmitting, the single sending station creates a dissemination group that other interested receiving stations can join. TMPT organizes the receiving stations into a hierarchy of subgroups or domains (roughly corresponding to the subnets on which the stations reside) and the stations within the domain agree on a single *domain manager* that acts as a representative for all stations within the domain. Each domain manager is responsible for ensuring that all members of its domain (children) receive the transmissions successfully. TMPT uses NAK-based error control and uses the TTL (Time to Live)[3] field of a NAK packet to minimize the scope of the retransmission request. The authors note that the control tree is built solely on the

[3] See the discussion of the use of TTL in Section 2.2 of this thesis.

transport layer (separate from that within the multicast routers) and requires no explicit support from the multicast routers to operate.

One sender-initiated protocol that had a particularly important influence on the receiver-initiated protocols was the Negative Acknowledgment with Periodic Polling (NAPP) protocol introduced by Ramakrishnan [RAM87]. The NAPP protocol was introduced as a broadcast protocol for LANs. In NAPP, receivers detect missing packets through the use of sequence numbers and send a NAK packet to notify the senders of a loss. Because the loss of the last packet transmitted would go unnoticed by receivers, periodically the sender would poll each of the receiving stations and have them transmit the last sequence number received. This sequence number serves as an ACK for all packets prior to that sequence number and allows the sender to determine the state of the receiver set. NAPP was the first protocol to introduce NAK Suppression, a concept that was later used in XTP, SRM, TMPT and others. Since all transmissions are multicast, it is possible for stations to listen to the NAKs transmitted by other stations and refrain from sending a NAK that has already been sent by another member of the group. This principle has been widely used and forms the basis of the multicast structure used by SRM [FLO95] as well as the TMP protocol introduced in this thesis.

2.1.2. Receiver-Initiated Protocols

One of the main assumptions in sender-initiated protocols is that the sender's knowledge of the receiver set is a critical component that must be provided by the protocol. For distributed computing, this is clearly a reasonable assumption. Critical tasks might be

carried out on various stations in a distributed environment and the failure of any station as well as any breakdown of the communications network must be known to the other stations in the group. However, explicit knowledge of the receiver-set is not as critical for tele-collaborative applications.

A number of protocols have challenged the assumption that reliability requires that the sender must verify receipt at each receiver. In receiver-initiated protocols, the burden of reliability on each receiver. Although the sender has a far more limited knowledge of the receiver set, the overhead associated with the error correction mechanisms is greatly reduced. By placing the burden of ensuring reliability on the receivers, larger groups can be supported [PIN94]. The question of whether or not receiver-initiated protocols provide true "reliability" is not simple. It can be argued that unless the sender of the data knows that every station in the receiver set has received the data successfully the transmission is not reliable. Receiver-initiated protocols address reliability only from the receivers' point of view. The receiver-initiated protocols that we will discuss vary primarily in the structure placed on the group by the protocol as well as the means through which the retransmission of errors is communicated to the sender. With the exception of RMTP, all of the protocols discussed use NAK-based error correction.

The receiver-initiated protocol with the simplest group structure is the Scalable Reliable Multicast (SRM) Protocol [FLO95]. SRM is a highly decentralized protocol in which no explicit structure is used for forming a multicast group. As in NAPP, all transmissions are multicast to the entire group and NAK Suppression is used to prevent a NACK Implosion. Each station delays a random time after detecting missing data prior to requesting retransmission. As in XTP [STR92], stations damp their own request if they

detect that another station has requested the same data. If a station damps a request, an exponential backoff algorithm is used that doubles the delay and tests if other stations have requested the same packets. Periodically each station sends out a *session message* that contains information regarding the last known sequence number from each sending station. This enables receivers to determine if the last packet from that sender has been lost. The rate at which stations transmit these messages is dynamically adjusted based on the perceived size of the multicast group. The overhead associated with these messages is never allowed to exceed 5% of the bandwidth allocated to the group. SRM takes additional steps to enhance scalability by specifying that all stations store the previous transmissions from all other senders in the group. Any station can respond to a retransmission request for packets they have stored. SRM extends the concept of NAK Suppression, i.e., implementing the suppression of multiple responses to retransmission requests, by having stations delay a random amount of time before responding to a request and only responding if they detect no other responses to the request. Unlike the previous protocols mentioned, SRM guarantees no ordering of packets although order delivery from a single source can be trivially implemented with buffering at the receiver.

The Reliable Multicast Transport Protocol (RMTP) [LIN96] takes the tree-based structure introduced in TMPT and implements a receiver rather than sender-initiated protocol. The multicast group is organized in a multi-level hierarchical structure in which the receiver set is subdivided into local regions. A controlling station referred to as a *Designated Receiver* (DR) is then responsible for receiving ACKs from members in its region and then sending an ACK to the next higher DR. Because only DRs send ACKs directly to the sender, ACK implosion is avoided. DRs cache data so that repairs can be locally repaired whenever possible. A window-based flow control is implemented that requires

that the sender acknowledgment from the top level DRs before continuing to transmit data. Unlike TMTP [YAV95], RMTP requires a modification to the multicast routing software for the construction of the RMTP trees.

Another receiver-initiated protocol that introduces a structure onto the multicast group is the Log-Based Receiver-reliable Multicast Protocol (LBRM) [HOL95]. LBRM uses a hierarchy of *log servers* that store the data transmitted during the session. The protocol does not specify whether packets remain available at the log server for an indefinite period or whether packets are discarded to reduce memory requirements. This decision is left up to each individual application. Log servers are very similar to the role of a *designated receiver* (DR) in RMTP with the exception that a NAK rather than ACK-based error correction scheme is used. LBRM uses heartbeat messages that are very similar to session messages in SRM [FLO95]. Since sequence numbers are used to identify missing packets, the loss of the most recent packet from any sender could go undetected. Heartbeat messages containing the last sequence number transmitted by a sender allow receivers to detect the loss of these packets that might otherwise go undetected. In SRM, as the group size grows large, it becomes more and more possible for the most recently lost packets to go undetected for longer periods due to the infrequency of session messages. LBRM overcomes this problem by varying the heartbeat interval, not in relation to group size, but relative to the interval with which senders are sending data. LBRM introduces the term *freshness* to refer to the degree that a receiver's state is up to date with the source. LBRM ensures freshness by sending heartbeat messages more frequently for frequent senders.

2.2. Internet Multicast Backbone (MBone)

Up until the late 1980's, multicasting had not been possible over the Internet. In 1989, support for multicasting at the network layer was integrated into the Internet Protocol with the establishment of IP multicast addressing [DEE89]. This mode of addressing, combined with routing and the Internet Group Management Protocol (IGMP), developed by 1992 into what is today known as the *Internet Multicast Backbone* or MBone [CAS94]. Since then, this technology has gained significant momentum[4] and is a widely accepted standard for multicasting. The MBone and the Internet currently operate over a shared medium. In its current form, the MBone exists not as a stand-alone network but as a virtual network

[4] In the summer of 1996, Stardust Technologies joined with many Internet hardware vendors (3COM, Cisco, and Bay Networks just to name a few) to endorse and further the hardware standardization of the Mbone through the Multi-Vendor IP Multicast Initiative (http://www.stardust.com/multi/). Through this initiative the hardware community hopes to advance the use and hardware support of MBone technology.

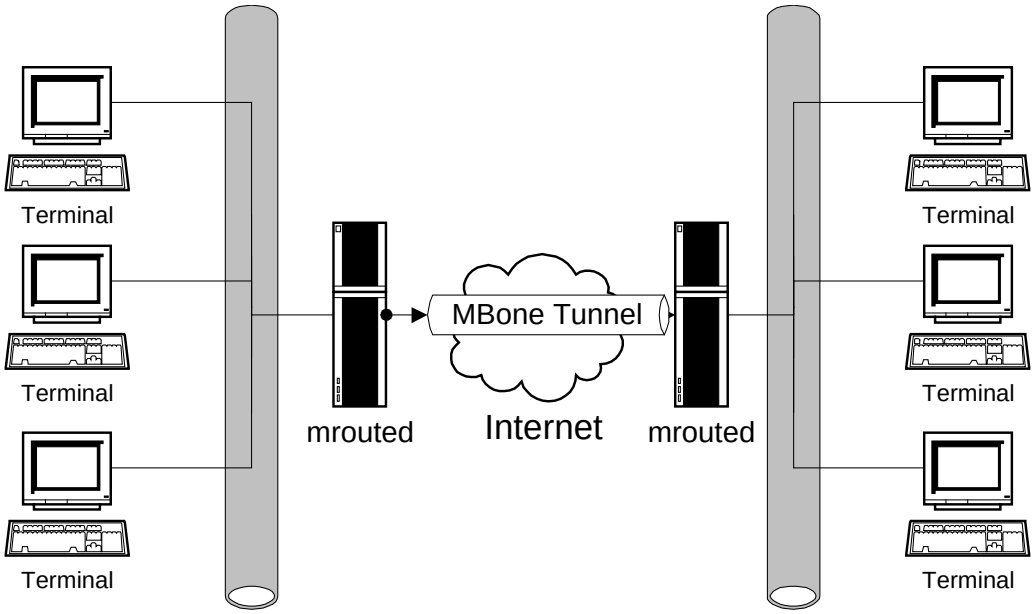


Figure 1 - Internet Multicast Backbone

consisting of routers that are capable of supporting multicast (*mrouters*). These mrouters are either workstations running the mroute software as a daemon process (*mrouted)*, or Internet routers that are multicast capable. Because the large majority of routers employed within the Internet are incapable of routing IP multicast datagrams, multicast packets are transmitted between workstations that run the mrouted daemon. This is accomplished by encapsulating multicast packets within regular IP packets and unicasting the encapsulated packets between multicast routers through *"tunnels"*. With the large number of major hardware vendors dedicated to supporting IP Multicasting, tunnels (as well as workstations running as mrouters) should gradually become unnecessary as the Internet backbone itself becomes multicast capable.

The MBone introduces two ways to control the distribution of multicast packets [MAC94]. The simplest method is by labeling each packet with a field known as the TTL (time-to-live). This field specifies the number of hops that a multicast packet is allowed to travel prior to be discarded by a multicast router. As the packet travels across the network, the multicast routers decrement this field and eventually discard the packet when the TTL value goes below a given threshold. The Internet Engineering Task Force (IETF) has imposed TTL conventions as shown in Table 1. The other method of limiting multicast packet distribution is through the use of sophisticated pruning algorithms to dynamically restrict the scope of multicast transmissions [WAI88]. These algorithms "prune" branches off the multicast routing tree where no members of the multicast group are attached.

TTL	Scope
0	Same Host
1	Same subnet
32	Same Site
64	Same Region
128	Same Continent
255	Unrestricted

Table 1 - IETF TTL Conventions

The primary multicast routing protocol used today in multicast routers is *DVMRP*, the *Distance-Vector Multicast Routing Protocol* [DEE91]. In DVMRP, routers maintain an entry in their routing tables for every subnet on the MBone. A delivery tree is built through the exchange of reverse path distance messages. Periodically each router exchanges messages

with each of its neighbors to update its routing table. Once the delivery tree is built, the multicast router uses Reverse Path Forwarding (RPF) [WAI88] to determine whether or not datagrams are forwarded to each of its neighbors (i.e., the datagram is forwarded if its neighbor is on the shortest path to the destination). This forwarding technique is then enhanced by on-demand "pruning" of tree branches not leading to group members[DIOT96]. Given the exponential grown of the MBone and the rapidly changing network topology, the processing costs of dynamically recalculating these routing tables have become unacceptably high. [THY95] predicts that limitations in the memory and/or processing resources at mrouters will cause MBone routing to collapse.

DVMRP suffers from a high memory demand to store the large flat routing tables as well as the processing costs incurred due to changes in the network topology. A number of new protocols have been introduced to overcome these problems . One method is through a hierarchical implementation of DVMRP in which the MBone is partitioned into a number of non-overlapping regions. Routing is then accomplished between the regions by what are called *Boundary Routers (L2 Routers)* using DVMRP. Routing internal to each region is done by *L1 Routers* and can be done using DVMRP or some other routing technique [THY95]. Because each individual routing table contains only a subset of the entire MBone topology, the size of the routing tables (and therefore the memory requirements at the mrouters) is significantly decreased. Furthermore since the partitions isolate the L1 Routers from changes to the MBone topology in remote regions, only a few routing tables must be changed in response to topology changes rather than every router in the network.

Another method proposed by the Open Shortest Path Working Group is known as *MOSPF* [MOY94]. This protocol is based mainly on an extension of a link state routing protocol and uses the existing the Link State database. Using this database, routers form efficient "source-based trees" or a "shortest-path tree" without the need to flood the network. Because the routing is accomplished using both the source and destination groups, datagrams intended for destinations beyond the reach of the TTL can be immediately discarded, improving link efficiency. There have, however, been shown to be significant disadvantages to this protocol in its inability to support heavy loads efficiently. MOSPF implementations must compute a new source-based tree for each source-group combination on demand [DIOT96]. The computations needed to create these trees can become computationally expensive for large numbers of groups with multiple senders.

A third alternative to DVMRP is *Protocol Independent Multicast (PIM)* [DEE95]. PIM seeks to overcome the scalability difficulties inherent with both DVMRP and MOSPF by augmenting DVMRP with additional operating modes to support *sparse (SM)* or *dense (DM)* groups. Sparse groups (SM) are characterized by group membership that is spread out thinly across regions of the Internet with the number of members in any given region being relatively small. By contrast, dense groups (DM) are characterized by a large number of members within a relatively confined portion of the Internet. PIM provides different services to differently characterized groups by forwarding data on all outgoing interfaces until pruning occurs in the case of dense groups[5] and forwarding data only on those interfaces from which explicit join messages have been received in the case of sparse groups. In its current state, the multicast routers that make up the MBone are

[5] Essentially operating in the same way as DVMRP except that the unicast routes are imported from existing unicast tables rather than incorporating a unicast algorithm in the specification and implementation directly [DIO96].

implementing a mix of all three protocols (DVMRP, MOSPF, and PIM). Since DVMRP is the primary routing protocol, non-DVMRP routers must implement a sufficient subset of DVMRP to "fool" the remaining routers into performing predictably [THY95].

Group Membership on the MBone is implemented using *the Internet Group Membership Protocol* (IGMP) [DEE89]. Multicast addresses are different from the usual IP address that identifies all stations on the Internet. Each IP address maps to a physical location and a specific device. Multicast IP group addresses are logical and do not map to a physical location but rather are bound dynamically to a set of local network interfaces on a set of IP networks. Again, it is important to contrast this with being bound to a set of physical addresses. Multicast routers maintain a list only of the other subnets that have stations belonging to the group. No list of physical addresses is needed or maintained. IGMP is used by stations to indicate their desire to participate in a multicast group. They do this by transmitting an IGMP control message to the multicast router on their local subnet. IGMP is also used amongst routers to report host group membership. IGMP places no restrictions upon who may or may not join these groups. RFC 1112 merely defines two operations *"JoinHostGroup"* and *"LeaveHostGroup"* as extensions to the IP Service Interface. There is no support within current versions of IGMP for access control lists or other restrictive measures on membership.

It is important to keep in mind that multicast IP is an inherently unreliable datagram protocol. It provides a "best effort" datagram service with no additional overhead to ensure reliable or ordered delivery. To provide reliability for packets that are transmitted with multicast IP, reliability protocols such as those introduced in Section 2.1 must be implemented.

The most popular applications in use on the MBone today involve the transmission of real-time audio and video data. The Real-Time Transport Protocol (RTP) is the accepted standard for real-time data and is discussed in the following section.

2.3. Real- Time Transport Protocol (RTP)

The Real-Time Transport Protocol (RTP) was designed to provide end-to-end delivery services for data with real-time characteristics [SCH96]. Specifically, RTP was introduced to meet the demands of the video and audio conferencing tools designed for the MBone. Real-time applications desire network support that provides bounded delay, an acceptably low rate of error, bounded jitter and some guaranteed throughput [LIE95]. RTP does not provide any quality of service (QoS) guarantees. Rather, RTP provides the feedback mechanisms so that delivery of this real-time data such as video and audio can be optimized despite the inevitable network variations present in the MBone.

RTP identifies each source in a multicast group by a unique identifier called a *Synchronization Source (SSRC)*. This identifier is independent of the IP address so that more than one process on a single IP address can be active in an RTP session. Because the algorithm used to choose an SSRC includes the possibility of two stations choosing identical SSRCs, RTP implementations must include the means to detect and resolve these conflicts. RTP introduces a second field known as a *Canonical End Name (CNAME)* in the format of *user@host* that is used to provide a unique identifier through which duplicate SSRCs can be detected and resolved.

The RTP control protocol (RTCP) was designed to monitor the quality of service and to convey information regarding group membership. This information can subsequently be used by applications to implement dynamic flow and congestion control. Because MBone multimedia sessions have grown large[6] and all stations in the group send these control messages, a flow control mechanism is applied that prevents the overhead associated with RTCP packets from exceeding 5% of available bandwidth. The primary RTCP packet types that carry out these functions are the Receiver Report (RR) and the Sender Report (SR) [SCH96]. SRs are transmitted only by those stations actively sending data since the last time that station sent out a SR or RR. Stations not transmitting in the last interval send out RRs even if they have previously been an active sender. Both contain information regarding the quality of service from the senders in the session including the fraction of packets lost, cumulative number of packets lost, interarrival jitter, and two timestamps that enable estimates of round-trip delivery latency. The SR includes only five additional fields about the sending station. The information in these reports can be useful not only for the sender for the implementation of dynamic flow control but also for other receivers or external monitors [SCH96]. Receivers can determine whether transmission problems are local or global and it is even possible for an external network monitor to use the contents of RTCP packets to evaluate the multicast performance of the network.

By providing the mechanisms that enable applications to implement dynamic flow and congestion control as well as group membership, RTP has become a standard for the multicast distribution of real-time data over the Internet. We have chosen to use the RTP data packet format as well as the RTCP protocol as the basis for the TMP protocol. By

[6] Recent sessions of video and audio transmitted from space shuttle missions have exceeded 200 participants.

using a well accepted packet format and control structure, we hope to be able to take advantage of ongoing research into using RTCP for dynamic flow control and quality of service (QoS) monitoring [BUS96] and apply the results to reliable multicast.

3. A New Approach to Reliable Multicasting

Existing reliable multicast protocols, although appropriate for individual applications, fall short of addressing the complete range of ways in which tele-collaborative applications vary in their requirements. In particular, many existing protocols [CHA84][BIR87] were designed to meet the stringent requirements of distributed processing rather than the more relaxed assumptions that are possible with tele-collaborative applications. Consider a multicast application that allows participants to jointly annotate and view a "whiteboard" [FLO95]. For this application the protocol need only provide a reliable service in which all packets eventually reach all participants. If used as a means for conducting a tele-lecture (a large number of receivers and a small number of senders), it is not critical to the sender to know immediately if a station is missing data or has dropped out. It is far more important to ensure reliability from the receiver's point of view as discussed in Section 2.1.2.

Our approach to providing a reliable multicast service differs from previous approaches in two fundamental ways. First, we believe that a protocol should complement the software development process. The application programmer should be isolated as much as possible from the complexities involved with ensuring reliability. The traditional argument against this approach is that efficiency cannot be maintained. We believe that through careful design and the introduction of tuning parameters, a single protocol can meet the needs of a wide range of applications. The second approach involves making a deliberate decision to support only those services that can be efficiently supported by a

receiver-initiated protocol. Based on previous evaluations of sender and receiver initiated protocols [PIN94] [LEV96], we believe that a receiver-initiated protocol is far more appropriate for efficiently supporting large tele-collaborative groups. However, receiver-initiated protocols simply cannot provide certain services that form the basis of many sender-initiated protocols. For example, as discussed in Section 2.1.1, many sender-initiated reliable multicast protocols introduce significant overhead to support a total and causal ordering of messages. Total and causal ordering is not easily obtained with a receiver-initiated protocol. Furthermore, although that level of ordering service is required for a distributed processing, it is unnecessary for most tele-collaborative applications where ordering may only be required from each sender. In the following sections, we will define a framework or *design space* that clearly defines the range of services that applications require and discuss in detail the problems associated with providing those services. We will conclude by discussing a number of important protocols and how they support the range of services defined by the design space model.

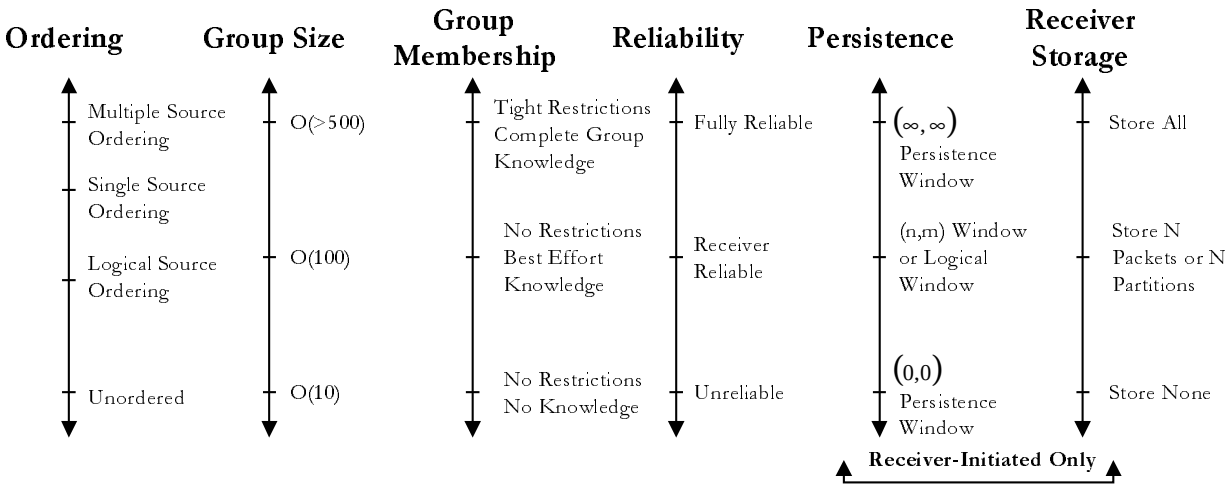


Figure 2 - A Reliable Multicast Design Space Model

3.1. A Reliable Multicast Design Space

As shown in our reliable multicast design space(Figure 2), we have identified five basic variables in both reliable multicast protocol design and in the demands that specific applications place on reliable multicast protocols. The dimensions of the design space include *ordering*, *group size*, *group membership*, *reliability*, *persistence*, and *receiver storage*. Each variable will be discussed in detail in follow-on sections. With our design space, we can graphically depict the *service ranges* provided by any multicast protocol by indicating the range of each variable for which the protocol is applicable. Similarly, the demands that an application places on a reliable multicast protocol define a line through each of the scales indicating the particular *requirements points* of each service variable required by the application. A protocol is deemed appropriate for a given application when the service ranges provided by that protocol fully encompass the requirements points as defined by the application. Previous reliable multicast protocols [WHE94] [STR92] [ARM92] addressed primarily the

possible ordering requirements and the impact of varying group sizes. However, applications vary far more in the demands they place on delivery protocols than just supporting varying levels of ordering and support for large group sizes.

It has been widely argued [FLO95][CLA90] that the requirements of different multicast applications are far too diverse and a single transport protocol cannot provide an appropriate service for a wide range of applications. Further, it has been argued [FLO95] that providing a protocol that meets worst-case requirements, e.g., total ordered delivery from multiple sources, would introduce substantial and unnecessary overhead for applications that have less stringent ordering requirements. We fundamentally disagree with these assumptions.

We approach the design of a reliable multicast protocol from the standpoint of the application developer. Although the efficiency of a protocol is of great importance, the benefits gained in terms of software development by implementing a single reusable protocol far outweigh any efficiency lost. We argue that for the effective development of multicast applications, a more flexible protocol is needed that can efficiently adapt the services provided to meet the needs of a wider range of applications. Instead of placing a significant burden on the application programmer by integrating many of the complex algorithms used in reliable multicast into the application, we propose the Tunable Multicast Protocol (TMP) that enables the programmer to set a number of parameters specifying the services provided by the protocol. TMP is designed specifically to meet the needs of tele-collaborative applications rather than the more stringent requirements of distributed computing.

In the next section, we will discuss each the significance of each dimension of the reliable multicast design space in greater detail as well as discuss the range of each dimension addressed by our new protocol.

3.1.1. Ordering

A multicast protocol should provide incoming data to the application in an order that is appropriate for the application rather than a default transport service. We discuss two examples of applications with very different requirements. To demonstrate the range of services required by a multi-purpose protocol, consider the requirements of real-time applications providing video or audio streams. Data must be presented to the application in a timely manner without regard for reliability. Losses are expected, causing only a minimal degradation in the playback quality. For this type of application, it is desired that a multicast protocol provide an unordered and unreliable service. At the opposite end of the spectrum of required ordering services, consider the example of a multicast plant automation application. A plant automation system would likely consist of a number of sensors transmitting data back to one or more controllers. For this class of application, a total ordering of data is imperative. Not only is it important that the transmissions from a single sensor be delivered in the order that they are transmitted, the relative ordering of transmissions from multiple sensors might be critical to the performance of the system. For example, given two sensors A and B, it might be important to the controller whether sensor A sent a message before or after sensor B sent a message.

Ordering has long been at the core of reliable multicast protocols and has its roots in distributed systems. [BIR87] first proposed *causal ordering*[7] of messages, based largely on Lamport's system of logical clocks [LAM78], and implemented it in the ISIS toolkit [BIR88][BIR91]. [GAR91] introduced three possible ordering properties, *single source ordering*, *multiple source ordering*, and *multiple group ordering* in addition to strict causal ordering. Multiple group ordering addresses the ordering of data belonging to multiple groups. This level of ordering is beyond the scope of this discussion. In order to more completely address the possible range of ordering services, we consider the following four ordering services:

1. Multiple Source Ordering - This service guarantees a total and causal ordering of all data. This level of service is by far the most demanding on the transport protocol and requires that group members be able to distinguish between the logical (sequence number based) or temporal (unique global timing) relationship between all data transmitted by members of the group.
2. Logical Single Source Ordering - With this method of ordering, each source labels its transmissions with a logical tag indicating a logical grouping of outgoing data. Ordering is provided independently for each unique source (the SSRC in the RTP protocol) for each logical grouping of outgoing data. No other guarantees are made on the ordering of data relative to the data of other group members or relative to other data that is part of a different logical group transmitted from the same station.
3. Single Source Ordering - This case is a simplified version of Logical Source-Based Ordering in which only a single logical grouping of data is allowed per source. A single source ordered service delivers data in order from each source in the group. Again no guarantee is made on the ordering of data relative to other members of the group.
4. Unordered - In this mode, data is immediately presented to the application upon receipt. No ordering services are provided. This service is appropriate as

[7] Discussed in Section 2.1.1.

mentioned for real-time applications in which the timely delivery of data is preferred over ordering.

We have made a deliberate decision to limit the scope of the ordering services that are provided by TMP. The case of multiple source ordering is by far the most demanding service to provide and is clearly not easily provided by receiver-initiated protocols. Single source as well as logical source ordering can be easily implemented through the use of sequence numbers applied at each source Existing protocols implementing multiple source ordering have been shown to be inefficient since multiple messages must be exchanged for the transmission of a single data packet. In the following section we discuss how global ordering services as well as other factors affect large groups.

3.1.2. Group Size

The task of designing a reliable protocol to support tele-collaboration for very small groups (2-5) is trivial. Since efficiency is not critical for small group sizes, the overhead introduced by even the most simple acknowledgment based scheme for reliability is relatively small and inconsequential. For larger groups, the overhead needed to provide reliability grows considerably. The central issue in efficiently supporting reliable group communication is the minimization of total message overhead [RAJ92]. The overhead that a protocol introduces is directly related to the services that protocol provides. For example, it is clear from our discussion of existing reliable multicast protocols that sender initiated protocols providing total ordering require far greater overhead to provide reliability than receiver initiated protocols[LEV96]. Similarly, any protocol that implements strict group

membership procedures must communicate that group membership information amongst group members to ensure consistency.

There is a positive correlation between the ordering services provided by a protocol and the group sizes that protocol can efficiently support. Most reliable multicast protocols that seek to provide a total and causal ordering service [CHA84] [BIR88] [LIN96] require that a central controlling station be used to establish a global ordering of messages. Communicating through a central station has a negative impact on these protocol's ability to support large groups in two ways. First, prior to sending a broadcast to the entire group, a station must negotiate all transfers with the central station. Because of this negotiation, these protocols require anywhere between 2 and 3 messages per broadcast[8]. Secondly, to respond robustly in the case of a failure of the controlling station, each protocol must renegotiate among the members of the group to identify and agree on a new controlling station.

Receiver-initiated protocols can support large group sizes since the additional overhead required to provide total ordering and maintain a consistent view of group membership is eliminated. Although the SRM protocol claims to be scalable to several hundred participants, it nonetheless experiences significant overhead proportional to group size by transmitting periodic "session messages" [FLO95]. SRM controls the bandwidth consumed by session messages by limiting the percentage of bandwidth that these messages are allowed to consume. Specifically, SRM dynamically adjusts the rate at which these messages are transmitted in response to changes in group size. Although the authors of

[8] This number varies due to "token" passing schemes when distributing the control overhead among multiple stations.

SRM claim that this adjustment allows SRM to support extremely large group sizes [FLO95], it is important to keep in mind that as the frequency of these "session messages" decreases, so too does the responsiveness of the protocol to losses identified through those messages.

TMP's approach to supporting large groups is very similar to that of SRM. However, we have modified the dynamic algorithm used to transmit the information in SRM's "session messages" so that the performance of the reliability mechanism does not degrade with larger group sizes. This algorithm is discussed in detail in Section 4.2.3.

3.1.3. Group Membership

The mapping of group members to an underlying multicast protocol defines a "*network group*" [DIO96]. The efficient management of network groups is critical to the performance of any multicast protocol. It has been argued [LEV96] [PIN94] , that the overriding factor that determines the scalability of a reliable multicast protocol is not necessarily in the mechanism used to ensure the reliability of data but whether or not each sender keeps track of the state of the receiver set (group membership). Group membership is therefore critical to any reliable multicast protocol and is definitely related to the size of a group that protocol can reasonably support.

As discussed in 2.2, the Internet Group Membership Protocol (IGMP) is the most widely used multicast group management mechanism in use today. Since IGMP does not provide restrictions on group membership, any multicast protocol desiring to provide a group membership service must do so explicitly. Group membership protocols may

provide restrictive access controls or merely attempt to provide a view of group membership for verifying the delivery of data. The task, however, of maintaining a consistent view of group membership in the face of failure is non-trivial. Any truly useful protocol must be robust to station failure and ensure that there is a consensus on the membership of the group despite the dynamics of stations failing, joining, and leaving. The problem of group membership for asynchronous systems has been the subject of intense investigation [HIL95] [KAA91] [MIS91] [MEL94] [RIC91]. Any solution should meet the following criteria [CHT96]:

1. The view of membership must be weak enough that a consensus can be reached among group members. If no consensus is possible, it is said to be unsolvable.
2. The algorithm must be strong enough to be useful in designing fault-tolerant distributed applications.

In [CHT96], the authors argue that the two criteria above are incompatible with asynchronous systems with failures and that, under certain assumptions, there is no solution to this problem. It is possible in any system for stations to fail without notifying other members. The MTP Protocol [FRMA90] recognizes this problem and has implemented a "fuzzy" knowledge of group membership in which stations accept that there may be members that have crashed or excluded even though they have not crashed. Given the difficulty of maintaining a coherent view of group membership in the face of station failure, the fault tolerance of any protocol that assumes a consistent view of group membership amongst participants must be closely scrutinized.

Since TMP is a receiver-initiated protocol, any attempt at maintaining a consistent view of group membership would be inconsistent with the basic concept of receiver

reliability. Instead, TMP uses the group membership reporting primitives of RTP to provide a "best effort" reporting of group membership but makes no guarantees of consistency of group membership across all members of the group.

3.1.4. Reliability

Given that a consistent view of group membership is difficult, if not impossible, to maintain, a re-examination of the entire concept and semantics of reliability is useful. The traditional definition of reliability requires the following properties [SEG82]:

1. *Completeness*: Multicast messages are delivered to each destination in the same order as sent by the source, without message duplication or loss.
2. *Finiteness*: Each multicast message is accepted by all destinations in a finite amount of time after its release at the source.

Two additional properties may be introduced to further refine the possible services provided [BIR91]:

3. *Atomicity*: Either all stations in the group receive a transmission or no station receives it (in the case that the sender fails).
4. *Causality*: When two transmissions are transmitted (by possibly two distinct stations) *m* and *m'*, every station receives *m* and *m'* in the same order as they were transmitted.

These properties are not trivial to guarantee. As an example, consider what must be known to guarantee *Finiteness*. The requirement that *all* stations receive a packet implies the requirement that a common view of group membership be maintained. Similarly, in order to guarantee *Atomicity*, a similar common view of group membership is required. As we discussed in Section 3.1.3, a consistent view of group membership is difficult, at best. to

maintain. Furthermore, unless the underlying network is reliable no assumptions can be made on the reliability of a multicast protocol. Specifically, even the most robust reliable multicast protocol is subject to network partitioning and failure.

Other classes of protocols seek to define reliability from the point of view of the receiver. In these protocols, senders do not attempt to maintain an agreed upon view of the receiver set. Finiteness and atomicity guarantees are clearly not possible. In order to correct errors in transmission, the receiver must only have the ability to detect losses and take action to ensure delivery. Only from the receiver's perspective, can *Completeness* be guaranteed.

Given the difficulties in meeting the strict traditional definitions of reliability, we introduce the following levels of reliability:

1. *(Fully) Reliable* - A fully reliable protocol must meet the conditions of Completeness, Finiteness, Atomicity, and Causality as described above. This level of service is normally associated with a totally ordered, reliable, atomic multicast service. This is the standard level of service that most reliable multicast services desire to provide [CHA84] [BIR91] [STR92] [MON94].
2. *Receiver Reliable* - In a receiver reliable service, there is no requirement for a sender to have knowledge of the receiver set. The responsibility for ensuring reliable delivery is at each receiver [PIN94]. Generally, in receiver reliable protocols, receivers only inform a sender if an error is detected by transmitting a negative acknowledgement (NAK).
3. *Unreliable* - In an unreliable service, no attempt is made to retransmit missing or corrupted data.

While it may be argued that a *Receiver Reliable* service does not meet the reliability requirements of applications, receiver-initiated protocols have demonstrated better

scalability than protocols that require senders to maintain the state of the receiver set [PIN94]. Receiver Reliable protocols are not without drawbacks. Regardless of whether or not the sender tracks the state of the receiver set, it is up to the sender to determine when to deallocate the memory used to store packets that have been transmitted to the group. In SRM each sender stores all data for possible retransmission to correct errors and to bring new members up to date [FLO95]. Not only does the original sender store transmissions indefinitely, all receivers also store all packets received. Therefore, the overhead involved with bringing new members up to date as well as correcting errors can be distributed amongst the entire group.

In TMP, we provide for receiver reliable or unreliable delivery. Full reliability is compromised in favor of more efficient reliability modes. A detailed discussion TMP's reliability modes can be found in Section 4.

It has been shown that the requirement for infinite storage in some receiver-initiated protocols can lead to deadlocks [LEV96]. However, if we relax the assumption of infinite storage, we must address the following question: How long should senders or receivers store data? We address the methods used to determine when to discard packets in the following discussions of Persistence and Receiver Storage.

3.1.5. Persistence

Packets must be available for retransmission to accommodate error correction. If packets are discarded before error correction is performed, reliability is not possible. Similarly, packets must be available if new members that join the group are to receive a

"history" to bring them up to the current state of the session. Packets can be stored at the sender or at some designated centralized station as in LBRM [HOL95]. We have borrowed the term *Persistence* from the field of database research with a similar but not identical interpretation. Persistent objects in databases are those objects that are continue to exist after program termination [ELM94]. We define *Persistence* not in the sense of a lifetime beyond the bounds of program execution but as the lifetime of data within the duration of a multicast session.

It is important to note that *Persistence* is not applicable to sender-initiated reliable multicast protocols. In sender-initiated protocols [CHA84] [MON94], data can be discarded once it has been received by all members of the multicast group. Since the sender tracks the state of the receiver set, there is no requirement to store data beyond the time necessary to verify receipt at each receiver. The only consideration of persistence for sender-initiated protocols is the requirement to archive data so that some amount of packet "history" can be transmitted to new members to recreate the current state of the session. Note that the amount of data transmitted to new members must be specified explicitly by each sender or possibly specified by a central controlling entity for any particular application.

Receiver-reliable protocols must treat the issue of persistence differently. Since there is no explicit tracking of the receiver set, it is not possible for a sender to determine when all members of the group have received a particular packet. This forces the sender to either "guess" when it is safe to discard the data and risk being unable to respond to a valid request for retransmission or store all transmissions for an indefinite period. In the case of the SRM protocol, all data sent by a station (as well as the data received from all other

senders) must be stored indefinitely [FLO95]. Clearly, from our previous discussion of reliability, this introduces the drawback of infinite storage requirements. We define three modes of bounded persistence based on the criteria that each sender applies to determine whether or not to store the data:

- **Temporal Persistence** - Data is stored for error correction for a specific period of time and then discarded. New members receive all data transmitted within the last *t* units of time. The amount of storage required depends on the rate of transmission.
- **Spatial Persistence** - All data is labeled with a sequence number that establishes ordering. The last *n* packets transmitted are stored. Likewise, new members receive the last *n* packets transmitted to bring them up to date. The amount of storage depends on the maximum size of a packet.
- **Logical Persistence** - Data is partitioned by the application such that the last *n* partitions are stored. For example, in the case of a shared whiteboard application, the last *n* pages created could be stored. Similarly, for a multicast file distribution application, the last *n* files would be stored. Unless the application explicitly bounds the size of a partition, partitions and storage can grow arbitrarily large. New members can receive any number of partitions of data to bring them up to date.

MTP [FRE90] was one of the first protocols to address the issue of persistence. As we discussed in Section 2.1.1, MTP implemented temporal persistence, retaining packets for a specific period of time. A primary drawback of this method of determining persistence is that the availability for repair is related only to the time since the packet was generated. For applications in which data transmissions are infrequent but critical, this may requires data to be stored for long periods and introduce large storage requirements. If data is not stored long enough, the protocol will not be able to repair errors effectively.

Spatial persistence has similar drawbacks to temporal persistence. Although the storage requirements at the sender are bounded, it is clearly possible for reliability to be compromised under certain conditions. Particularly in a protocol with significant latency associated with error recovery, a station that is sending many packets over a relatively short period of time may not be able to correct errors. Packets may be quickly discarded without the opportunity to correct errors because of the rapidly advancing sequence numbers. To ensure that error correction can occur, either a large number of packets must be stored or restrictive rate control must be applied so that no sender can quickly advance sequence numbers beyond the protocols ability to correct errors. If used as a means to bring new members up to date, sequence number-based persistence can result in considerable inefficiency. Consider the case of a new member joining as a large file is being transmitted to the multicast session. If the new member receives only a portion of the file, it may be unusable. Similarly the new member may receive a large number of packets that precede the file that are no longer useful.

Logical persistence corrects a number of the inherent difficulties with both temporal and sequence number-based persistence. By allowing the application to specify which data should be stored for error correction, the risk of pertinent data being unavailable to correct errors is greatly reduced. However, the burden on the programmer is increased since the application must explicitly limit the size of any data partition. Without explicit partitioning, requirements for data storage may not be bound by the protocol. Logical partitioning has the added benefit of allowing the application to specify precisely what new members will receive to bring them up to date.

The TMP design supports both sequence number-based as well as logical persistence. The persistence services provided by TMP are discussed in Section 4.4.

3.1.6. Receiver Storage

One of the distinctive properties of SRM is that it requires not only original senders to store data indefinitely but also all receivers as well [FLO95]. The authors of SRM justify this requirement as necessary to distribute the load of error repair amongst the members of the group[9] as well as make it possible to implement "localized recovery". "Localized recovery" enables the members of the group "closest" to the point of the failure to respond most quickly to losses. To prevent an implosion of responses to requests for retransmission of missing data by multiple receivers that have the data in storage, SRM proposes a random delay in which stations verify that no other station has responded with the requested data and damping their response if they detect another station's response. Note that this is very similar to the NAK suppression algorithm discussed in Section 2.1.2. Although this approach can distribute the load associated with the retransmission of data, it has a number of drawbacks including,

- This scheme introduces two random delay periods before errors can be corrected. One delay period is introduced for NAK suppression at each receiver and the second prevents an implosion of responses to requests for retransmission. There is a clear design trade-off between the benefits of enabling multiple stations to respond to errors and the additional latency introduced.

[9] Referred to in [FLO95] as "exploiting the redundancy of multiple receivers".

- If all stations store all transmissions, this can cause a requirement for infinite storage at each receiver. Because each receiver must store all transmissions, SRM places a very high requirement on each station in terms of memory usage. In an application in which each station transmits with relatively similar frequency, the storage requirement is proportional to group size, an obvious drawback to supporting large groups.

- SRM refers to an algorithm in which stations delay based on what they estimate to be the round trip delay to the station requesting retransmission. In order to calculate round trip delay between stations without assuming a common clock, the periodic "session" messages must contain information on not only all senders in the group but also all receivers. [10] This requirement can cause the size of individual control packets to grow in proportion to the size of the receiver set.

In order to avoid some or all of these drawbacks, we define a range of possible storage options at each receiver to allow for maximum benefit from receiver storage without the disadvantages inherent in the SRM protocol. The range of values includes:

- (∞) - Receivers store everything transmitted. This requirement is identical to SRM and requires the introduction of a second random wait period as discussed above.

- (n) - Receivers store the last n packets or last n partitions of data (in the case of logical persistence windows). This allows for the distribution of repairs amongst the receiver set without the requirement for infinite storage. This requirement does not alleviate the requirement for the second random wait period.

- (0) - No receiver storage. This eliminates the second wait period but requires all data transmitted for error correction or to bring new members up to the current state to be transmitted from the original source of the data.

---

[10] The RTP specification gives examples of this calculation which must include a Δ parameter that is referred to in an RTCP Sender/Receiver Report as DLSR (delay since last sender report). This enables each sender to calculate the transmission time to each member of the group. In order to allow receivers to calculate their distances (and therefore benefit from "localized recovery", these reports must include a Δ parameter to indicate DLRR (delay since last receiver report).

In the TMP Protocol, we support the entire range of receiver storage and provide an additional mode in which no receivers store data to reduce error correction latency. TMP's receiver storage services are discussed in Section 4.2.2.

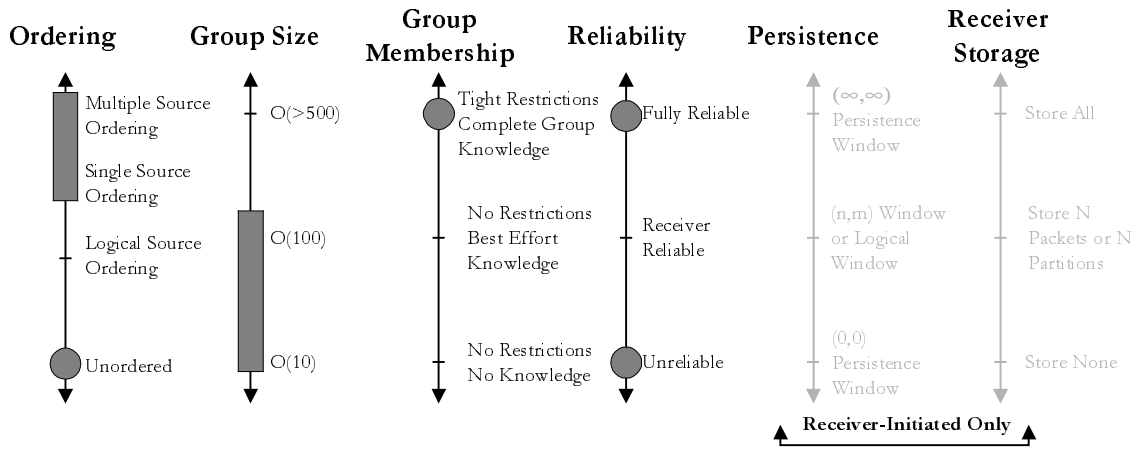


Figure 3 - RMP in the Reliable Multicast Design Space Model

3.2. Definition of Existing Protocols within the Design Space

The design space introduced in Section 3.1 serves as a metric by which we can measure the applicability and breadth of existing protocols. By defining existing protocols within the design space model, we are able to graphically depict the service provided by these protocols. Because of the differences between sender and receiver-initiated protocols, not all dimensions will be pertinent to all protocols.

Figure 3 shows the definition of RMP [WHE94] [MONT94] in the reliable multicast design space. As a ring based protocol, RMP keeps explicit track of group membership in order to create its ring structure. The group membership dimension is relaxed slightly

because of RMP's capability of allowing non-members to transmit to the group. RMP does make an attempt to remain flexible to a number of parameters including ordering and reliability. It allows for unordered or totally ordered delivery and provides for a reliability mechanism that allows for guarantees in which a subset of the group receive the data reliably (k-resilience) [WHE94]. Furthermore, RMP provides this range of services on a per-packet basis. As one of its design goals, RMP seeks to provide a flexible service that is appropriate to a wide range of applications. However, as a sender-initiated protocol, RMP does not address either persistence or receiver storage (displayed in gray).

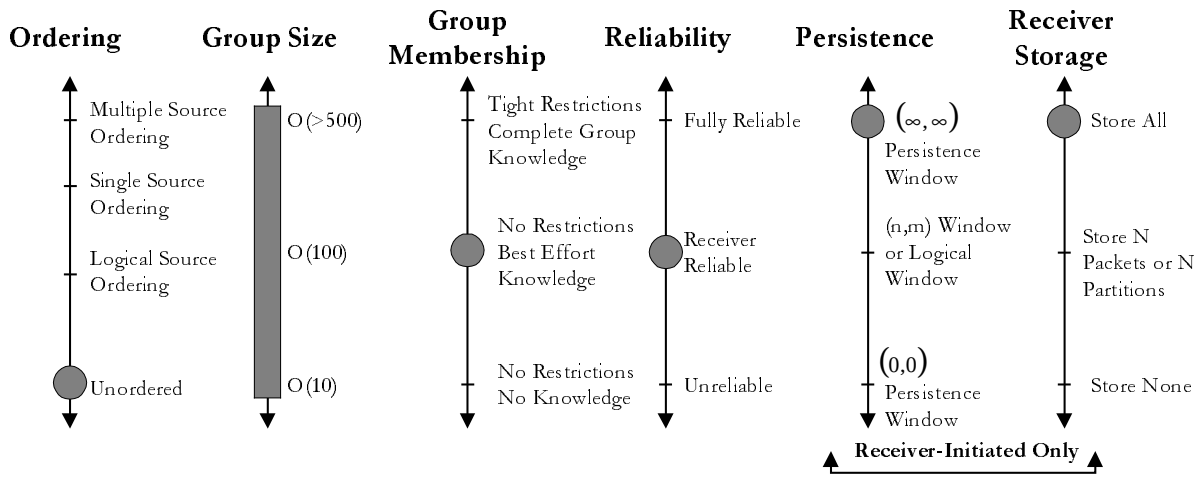


Figure 4 - SRM in the Reliable Multicast Design Space

SRM [FLO95], shown in Figure 4, addresses only a very narrow portion of the design space. This is not, however, surprising considering that the basic tenets of SRM were intended as a base service upon which other protocols could be based. SRM provides no ordering services, leaving the ordered delivery of data as an additional task for the application. The requirement for receivers to buffer all data to distribute the load of error correction can introduce a significant storage burden for many classes of applications.

In the following chapter, we introduce the Tunable Multicast Protocol (TMP). The TMP protocol was designed to maximize the portion of the design space addressed without introducing unacceptable overhead for applications requiring a subset of the services that TMP provides.

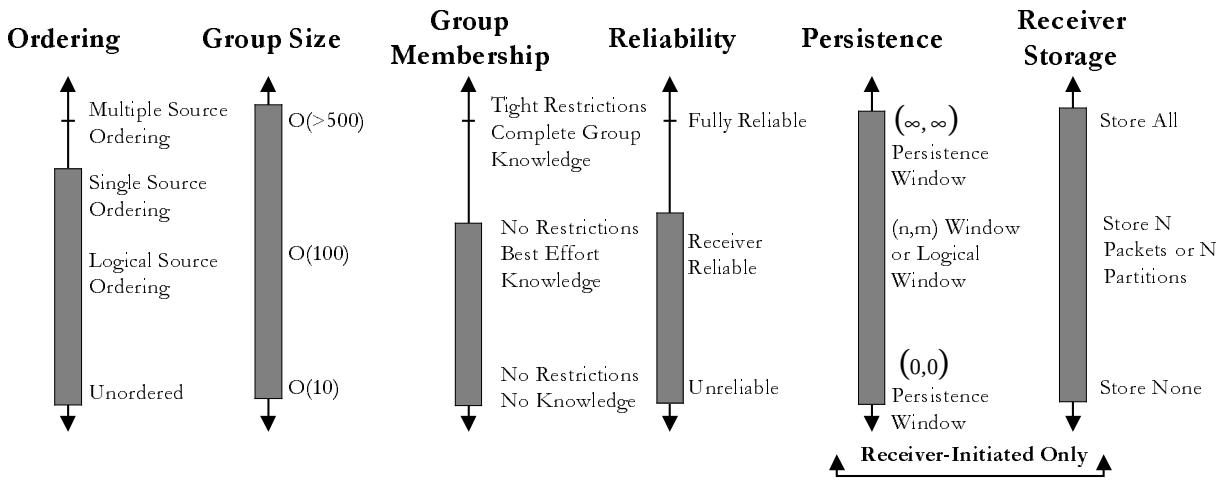


Figure 5 - Definition of TMP in the Design Space Model

4. The Tunable Multicast Protocol

As we have shown, existing protocols address only a subset of the multicast requirements of applications. We present a new protocol that can address a greater portion of the design space and therefore support a wider variety of tele-collaborative applications. The Tunable Multicast Protocol (TMP) is a receiver-initiated protocol similar to the SRM protocol presented in [FLO95]. TMP uses the packet format from the Real-Time Transport Protocol (RTP) discussed in Section 2.3. We address a larger portion of the design space by allowing the services that TMP provides to be specified through a number of *tuning parameters*. TMP is implemented as a reusable C++ class that applications can use to provide a reliable multicast service tuned specifically to the needs of their application. TMP addresses the following portion of the design space as shown in Figure 5:

- **Ordering** - TMP provides unordered and single source ordering as well as the logical single source ordering described in the Section 3.1.1.

- **Group Size** - The goal of TMP is to efficiently support group sizes approaching 500. This is a goal for the protocol. Verification of the performance of TMP with large group sizes is left to future work.

- **Group Membership** - TMP provides the kind of "best effort" membership reporting that is characteristic of RTP [SCH96] and places no restrictions on group membership.

- **Reliability** - As a receiver-initiated protocol, TMP provides receiver reliability. Any application desiring to receive explicit acknowledgment from the entire group can do so using the group membership reporting procedures mentioned above. These procedures will be described in detail in Section 4.1.

- **Persistence** - TMP implements the entire range of persistence services discussed in Section 3.1.5. Logical as well as sequence number-based persistence can be selected with any range of values. We will discuss these services in detail in Section 4.4.

- **Receiver Storage** - Each station within the multicast group can specify any value of receiver storage. For applications in which no receivers will store data (data is available from the source only), TMP provides a special mode which eliminates the delay that a station would normally execute prior to responding to requests for retransmissions of data. These modes are discussed in Section 4.2.2.

In Chapter 3, we described our new approach to multicasting tailored to the needs of tele-collaborative applications in which certain service guarantees are relaxed to provide a more flexible and efficient protocol. We introduce the TMP protocol as an example of this approach. We further argue that a single protocol can be designed that meets the needs of a wide variety of tele-collaborative applications. As we discussed in Section 3, there has been a great deal of argument directly against this notion. In previous work [CLA90] [FLO95], efficiency has been a primary concern. In particular, it has been implied that to maintain efficiency, protocols must be integrated with and customized for specific applications. We

believe that by marginally compromising efficiency with the introduction of tuning parameters, significant software engineering improvements can be realized.

The SRM protocol [FLO95] implements what is known as Application Layer Framing (ALF) [CLA90]. The primary goal of ALF is efficiency. ALF requires that the application, rather than a separate transport protocol, must frame data that is transmitted to the network. In ALF, packets are referred to as Application Data Units, or ADUs. ADUs remain in tact throughout the protocol processing. When ALF is combined with an orthogonal principle known as Integrated Layer Processing (ILP), efficiency is optimized since data is processed only once and is not copied [CLA90]. One drawback to this approach is that the application must not only define the framing of data to be passed between participants, it must also handle the complexities of error recovery rather than simply passing those ADU's to a reliable transport layer. Moreover, by integrating the transport protocol into the very fabric of the application, ALF and ILP place a significant burden on the application programmer making the goals of software engineering such as reusability and modularity far more difficult to achieve. The application is unable to abstract itself from the complexities of the underlying reliable communication protocol because the application is so closely integrated into the very framework of that protocol.

In TMP, the application is responsible for framing data (in accordance with ALF) but we reject the principles of Integrated Layer Processing (ILP). The application merely specifies the desired ordering and reliability guarantees and passes the data to the TMP protocol for reliable transmission. Rather than performing all processing in a single layer on one copy of the data, TMP performs a copy of the data between the application and transport layer. Without this copy, memory management at the application layer would be

unacceptably difficult. As an example of this complexity, consider the common case of an application sending a single data packet. In a sender-initiated protocol, the decision to discard the local copy of the packet is a simple one. The packet can be discarded once every receiver in the receiver set (or possibly a given station in an hierarchical protocol) has successfully received the packet. This decision is far more difficult in a receiver-initiated protocol since the sender does not track the state of the receivers. If only a single copy of the data is made, it is the application's responsibility to determine when it is appropriate to delete the single copy of the data. We have referred to the issue of when to delete data as persistence (discussed in Section 3.1.5). Forcing the application to deal with the complexity of Persistence is unacceptable to the software engineering process. By copying the data into the transport layer (and violating the principle of ILP), the application is free to discard data immediately upon transmission and let the transport layer determine when to delete its copy.

As discussed in Section 3, one of our primary goals was to design TMP such that its use would enhance the software engineering process. In [COH96], the authors define the results of software engineering as software that is reliable, understandable, cost effective, adaptable, and reusable. The TMP protocol was designed so that developers of multicast applications (particularly those in which reliability was desired) could have a "plug-in", reusable component that met the needs of their application. Furthermore, the protocol is flexible and efficient enough so that network and processing resources are not wasted on unnecessary and costly services. Since the reusability of any protocol is critical to its usefulness in designing and implementing applications, we determined that a C++ implementation would be preferable to a conventional C implementation. The modularity

of the code and the ability to present a simple object-oriented interface to a reliable multicast protocol makes C++ an obvious choice over conventional C.

In the following sections we will describe the specific services provided by TMP and the methods and algorithms used to achieve ordering, reliability, and persistence. In Section 4.5, we introduce the TMP application program interface (API). An overview of the underlying object model and algorithms are presented in Section 4.6. A detailed discussion of the packet formats are presented in Appendix A. It is important to note that not all services described in this chapter are fully implemented as of the date of this thesis. We will note where planned services are described that are not yet implemented.

4.1. TMP Ordering Services

One of the most significant design decisions made in TMP was to limit the range of ordering services provided so that total ordering is not supported. Based on existing analysis of sender and receiver-initiated protocols [PIN94], we determined that without a hierarchical approach, total ordering could not be efficiently provided for large groups. Furthermore, if the protocol did provide total ordering, the overhead introduced would unacceptably affect those applications not desiring that service. In TMP, all ordering is accomplished by assigning sequence numbers to each data packet. The application is not responsible for assigning sequence numbers to ADUs (the protocol assigns sequence numbers to all packets received from the application). Ordering services are specified by each station in the group during the start of that station's participation in the session. It is

not possible for stations to specify different ordering services for the same session. The ordering services provided by TMP are as follows:

- **Unordered** - Packets are delivered to the application immediately upon receipt regardless of order. The protocol will attempt to deliver every packet reliably without regard to the order of delivery. This service is appropriate for those applications in which immediate delivery is more important than ordered delivery. Unordered service is not currently implemented but can be trivially included.
- **Logical Source Ordering** - Packets are subdivided by the sending application into logical groups. Ordered delivery is guaranteed per sender by logical grouping. There is no guarantee on the ordering of packets from different logical groups.
- **Single Source Ordering** - Packets from each sender are delivered to the application in the order that they are presented to the transport layer for delivery.

If single source ordering is chosen as the desired ordering service, the protocol provides no support for logical grouping.

4.2. Error Correction

As we have mentioned, TMP uses unreliable multicast IP service for packet delivery. Multicast IP provides mechanisms for the detection of bit-errors and it discards corrupted packets. TMP's error correction mechanism must only detect missing packets and ensure retransmission of those packets. The RTP packet format includes a sequence number. However, since all outgoing packets (original transmissions and retransmissions) must have an increasing RTP sequence number, this sequence number is not appropriate to use as a means to ensure reliability. We have introduced a second sequence number, which we refer

to as a TMP Sequence Number. This sequence number is used to establish ordering only for original transmissions and is therefore used as the sequence through which reliability is ensured.

As with any sequence number-based reliability mechanism, the detection of lost packets can occur in two ways. First, by the receipt of a packet whose sequence number does not immediately follow the preceding packet, and second, through the use of "heartbeat messages" (see the discussion of LBRM in section 2.1.2) that identify the sequence number of the last packet transmitted from each sender. In Section 4.2.3, we will discuss the use of heartbeat messages in the TMP protocol.

TMP provides error correction through the use of negative acknowledgments (NAK). Receivers send NAKs for those packets that they determine to be missing and other stations retransmit only those packets that are explicitly requested (i.e., a *selective retransmission* policy). As in SRM [FLO95], all retransmissions as well as NAKs are multicast to the entire group. It is clear from our previous discussion of other multicast protocols that an ACK based scheme is inappropriate for a receiver-initiated protocol. A Go-back-n retransmission policy is also inappropriate since it would result in redundant multicast transmissions of packets, wasting valuable bandwidth. Each NAK, which we will refer to as a *repair request*, contains two sequence numbers. This enables multiple contiguous packets to be requested with a single repair request (reducing the overhead associated with error correction). Since NAK Suppression is used to prevent a NAK Implosion, we will discuss how each station ensures reliability in the following section.

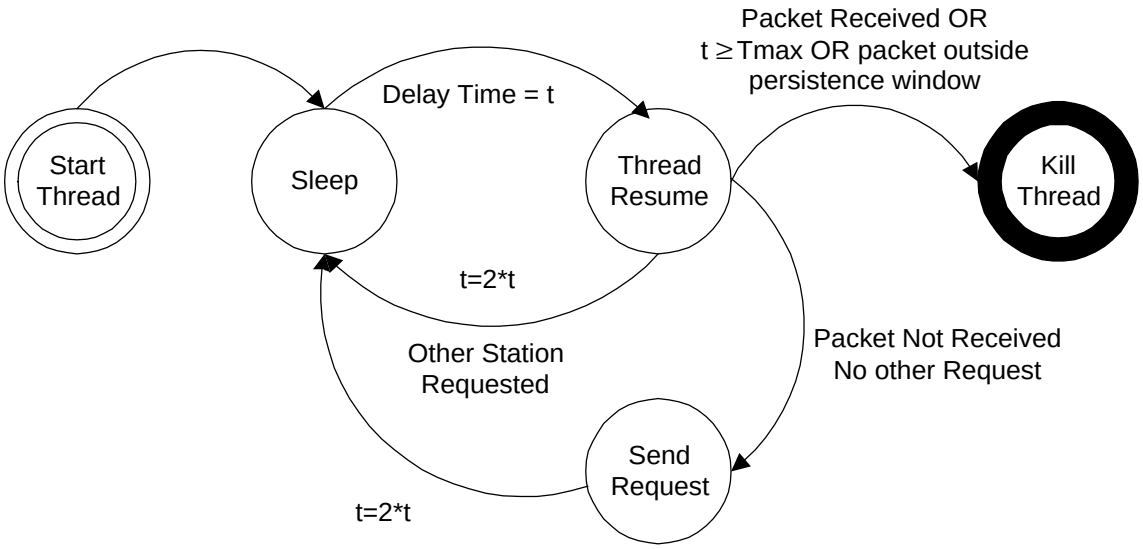


Figure 6 - Error Correction Thread Algorithm

4.2.1. NAK Suppression

Because receivers will multicast NAKs to the sender upon discovery of a loss, TMP could be subject to the NAK Implosion Problem [PIN94] if many receivers transmitted a repair request at the same time. We have adopted the NAK Suppression algorithm first introduced in [RAM87] and later used in the SRM Protocol [FLO95]. The implementation of this algorithm in TMP uses a multi-threaded approach and will be discussed as implemented in our protocol.

Once a loss is detected at a receiver (either through out of order sequence numbers or a heartbeat message), a thread is created at each receiver to recover the missing packet (or packets in the case of contiguous missing packets). Each thread shares the available processing resources at the station. If the protocol creates a large number of threads, the

system will "*thrash*", spending all of its processing resources switching between the threads. To prevent this from occurring, threads are not actually created immediately upon detection of a loss. We have placed a limit on the number of concurrent missing packet threads created by the protocol. If more threads than the limit are needed, the information regarding the loss is stored and queued (FCFS) for activation. Once a thread is activated, it remains active until the missing packets are recovered or the protocol determines that the packets are unavailable. The error correction thread algorithm is shown in Figure 6.

The thread's first action is to sleep for a random period of time. The formula for the random sleep period is given by $t = C_{min} + U[C_{Range}]$, where C_{min} denotes the minimum sleep time (in milliseconds) and $U[C_{Range}]$ denotes a uniform random variable between 0 and C_{Range}. This formula is a variation of the back-off timer values proposed in [FLO95]. During the sleep period, the thread monitors the repair requests of other stations in the group. When the thread resumes, it determines if the missing packets have been received. If so, the thread terminates. Otherwise, only if no other station has requested the packets does the station send a request for the missing packets. The thread then doubles the delay period and sleeps again. Figure 6 shows the ways in which the thread can terminate. The thread terminates if any of the following conditions occur:

- The missing packets are received. This is the "normal" termination mode.
- The sleep period reaches T_{max} and the packets have not been received. If this condition occurs, the packets are assumed to be unavailable. A special NAK_DENY message is sent to the application in lieu of the actual packet.
- New packets or partitions (in the case of logical persistence) are received such that the packet is known to be no longer available. Again a NAK_DENY message is sent to the application in lieu of the actual packet.

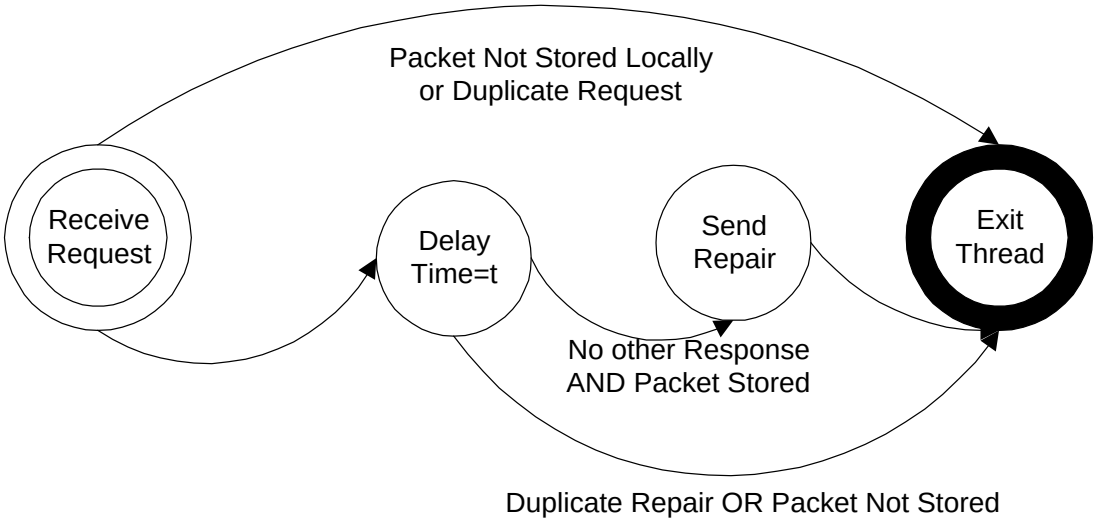


Figure 7 - Repair Response Thread Algorithm (Receiver Storage Mode)

The choice of T_{max} is left to the application developer as a tuning parameter. By stipulating a maximum time a thread will attempt to recover a missing packet, the thread is prevented from consuming processing time indefinitely due to a missing packet that can never be recovered.

4.2.2. Response Implosion Avoidance

If a protocol allows multiple stations to respond to a request for retransmission, it is possible for an implosion of responses to occur. Based on our discussion in Chapter 2, this is only a concern for receiver-initiated protocols such as SRM in which multiple receivers are able to respond to requests. As we discussed in Section 2.1.2, in the SRM protocol, each station delays prior to responding to a request for retransmission so that implosion is minimized. Note that this algorithm is very similar to the damping algorithm used by SRM for NAK suppression. A drawback to this approach is that this second delay period

increases the latency of error correction. For some applications, this increased latency may be unacceptable.

In TMP, we have defined two modes of repair depending on the application's sensitivity to error correction latency. The first mode is not fully implemented at this time but is defined here for completeness. In the first mode, all stations in the multicast session agree that only the original source of data responds to requests for retransmission. In this mode, the source of the data responds immediately to requests for retransmission. Repair latency is minimized and a response implosion is not possible since only a single station can respond to each request.

In the second mode, all stations have the option of storing packets transmitted by other stations[11] and responding to requests. The repair response algorithm is shown in Figure 7. TMP's algorithm is similar to that used by SRM to prevent response implosion [FLO95]. As in the case of the error correction algorithm, TMP uses a multiple threads to respond to requests for retransmission. Again, a thread limiting mechanism is employed to prevent too many threads from degrading performance. If insufficient threads are available, requests are queued FCFS and activated as threads become available. If the station does not have the requested packets in storage, the thread immediately terminates. If the packet is available, the thread sleeps for a random time given by $t = C_{min} + U[C_{Range}]$, where C_{min} denotes the minimum sleep time (in milliseconds) and $U[C_{Range}]$ denotes a uniform random variable between 0 and C_{Range}. This formula is identical to that given in Section 4.2.1 but the values of the constants (C_{min} and C_{Range}) need not be the same. Once the thread resumes, it

[11] We have made this optional to account for the difference in both network and processing capability of participants. Whether or not to do receiver storage and how much to store is a tuning parameter in TMP.

sends a retransmission only if no other station has sent a response. The thread then terminates since only the receiver is responsible for reliability..

These two modes of error correction enable TMP to provide flexible services depending on the tolerance of an application to repair latency. In our previous discussions, we have not specified how stations communicate requests for retransmissions of missing packets. In the following section, we will discuss how we modified RTP to provide error correction services.

4.2.3. Combining RTP with a Reliability Mechanism

TMP uses the Real-time Transport Protocol (RTP) for framing and for control. The goals of RTP seem to be contrary to the needs of reliable multicast protocols. RTP was designed to meet the needs of real-time applications such as video or audio streams. Real-time applications are far more concerned with the timely delivery of data than by its completeness. RTP and RTCP were designed specifically with those needs in mind. However, by adopting the RTP packet format as well as implementing its accompanying control protocol, RTCP, we inherit a framework through which quality of service (QoS) and receiver-set performance monitoring can be easily implemented [BUS96]. In this thesis, we introduce a new RTP profile for transmitting reliable data. The packet formats and descriptions are presented in Appendix A. The standard RTP data packet formats are used for both data and retransmissions. We implement a subset of RTCP and extended it to provide support for the reliable transmission of data. In the following discussion we will

introduce the problems associated with the use of RTCP Sender and Receiver Reports to achieve reliability and introduce an alternative solution.

The most obvious approach to using RTP for reliable multicasting would be to modify RTP's existing control protocol to ensure reliability. In fact, the implementation of TMP as of this thesis extends RTCP Sender and Receiver Reports to allow for error correction. However, we have identified drawbacks to this technique that make it necessary to introduce a different approach. In the RTP specification, the frequency of Receiver and Sender Reports is adjusted dynamically based on the size of the group. For large group sizes, this frequency can become very seldom as the total bandwidth consumed by these reports is set to 5% of the bandwidth dedicated to the session [SCH96]. In the SRM protocol [FLO95], a "session message" is defined that serves to keep the receiver set up to date with the current state of the sender in the absence of new data. In a very similar manner to RTP, the rate at which these messages are transmitted is varied according to group size in order to keep the overall bandwidth consumed by those reports low. Although not a factor in real-time systems, the long delay between sender and receiver reports can be a serious drawback if those report are modified so that they are means through which reliability is achieved.

To address the problem of long delays between Sender and Receiver Reports, we have adopted the concept of "heartbeat packets" first addressed by the LBRM Protocol [HOL95]. The LBRM protocol [HOL95] addressed the problems of long intervals between reliability messages with the introduction of variable rate heartbeat packets to ensure

"freshness".[12] We have adopted this concept for use in the TMP protocol and have introduced a new "application specific" RTCP packet in accordance with the provisions of RFC 1889 [SCH96]. We adopt the terminology of LBRM and refer to these control packets as "heartbeat packets". The complete packet format for the heartbeat messages is introduced and discussed in Appendix A. Different from SRM, heartbeat packets are transmitted only by stations that are the source of original data rather than by all members of the group. This allows TMP to scale more effectively in support of tele-collaborative sessions where only a small number of stations generate data and a large number of stations are "passive" receivers.

As in LBRM, the rate at which a sender transmits heartbeat messages varies depending on the time since the last packet was sent. LBRM was designed to support the needs of a distributed simulation environment with real-time requirements [HOL95]. In LBRM, the interval between heartbeat messages is started at a value h_{min} and is doubled whenever a heartbeat packet is sent, until it reaches a value of h_{max}. Each time a new data packet is sent, the interval is reset to h_{min}. The values chosen for h_{min} and h_{max} are given as h_{min} =.25 sec and h_{max}=32 sec. By choosing a small value of h_{min} and a large value of h_{max}, the LBRM protocol attempts to achieve a very short error recovery time while reducing the overhead associated with the heartbeat messages. In the TMP protocol, the same basic algorithm for modifying the heartbeat interval was chosen. However, because of the delays involved with NAK Suppression, the TMP protocol uses a higher value for h_{min}[13] and a much lower value for h_{max}. Clearly, any value chosen that was less than the average delay

[12] Recall from section 3 that LBRM introduced the term *freshness* to refer to the degree that a receiver's state is up to date with the source.

[13] Our experimental values chosen were h_{min}=1.5 sec. and h_{max}= 10 sec.

prior to requesting retransmissions of missing packets would result in unnecessary heartbeat messages. Also, a station will damp any heartbeat message if h_{min} has not elapsed since its last transmission.

Our second modification to RTCP to support reliability is to add a mechanism that enables stations to request retransmission of missing packets. TMP introduces a second "application specific" RTCP packet referred to as a *Repair Request*. The packet format for this packet is shown in Appendix A. Stations can request retransmission of any number of contiguous missing packets using this RTCP packet.

4.3. Rate Control

TMP uses a number rate control mechanisms. Rate control is critical to regulate the rate at which packets are transmitted to the multicast group. Rate control is important not only to limit the transmission of data so that buffers do not overflow, but also to limit the amount of control information so that control packets do not consume more bandwidth than is acceptable. We describe mechanisms that are applied both to control the volume of outgoing data and retransmissions as well as two dynamic mechanisms to control the overhead associated with the TMP control protocol.

TMP uses a "leaky bucket" rate control algorithm to control outgoing data. Leaky bucket flow control defines flow control through two parameters, a burst parameter and a rate parameter. In TMP, all packets transmitted on the data port of the RTP/RTCP socket pair [SCH96] are controlled through the leaky bucket mechanism. The application can

specify both a burst size and a throughput to tune the flow control for a given application (Section 4.5).

The dynamic rate control mechanisms that limit the amount of bandwidth consumed by control packets are described for completeness but are not yet implemented in TMP. RTP's mechanism to vary the frequency of Sender and Receiver Reports based on estimates of the size of the group will be implemented in TMP. However, this does not affect the frequency of heartbeat messages. Heartbeat messages are transmitted as described in Section 4.2.3. The total number of heartbeat messages as well as the number of repair requests is unconstrained regardless of group size. If heartbeat messages are transmitted strictly relative to the size of the group, losses may remain undetected for extended periods because of the infrequency of heartbeat messages. There is an implicit rate control mechanism on repair requests introduced by the NAK Suppression algorithm. However, we have not introduced any other rate control mechanism to limit the total percentage of bandwidth consumed by these messages. While this will limit the maximum size group supported by TMP, we consider the timely delivery of heartbeat messages to be critical to the error correction performance of the protocol. Although it is possible for an abundance of heartbeat messages to introduce network congestion for extremely large group sizes, the benefit of more timely error detection outweighs the overhead introduced. We believe further evaluation of the impact of heartbeat messages on total throughput is required.

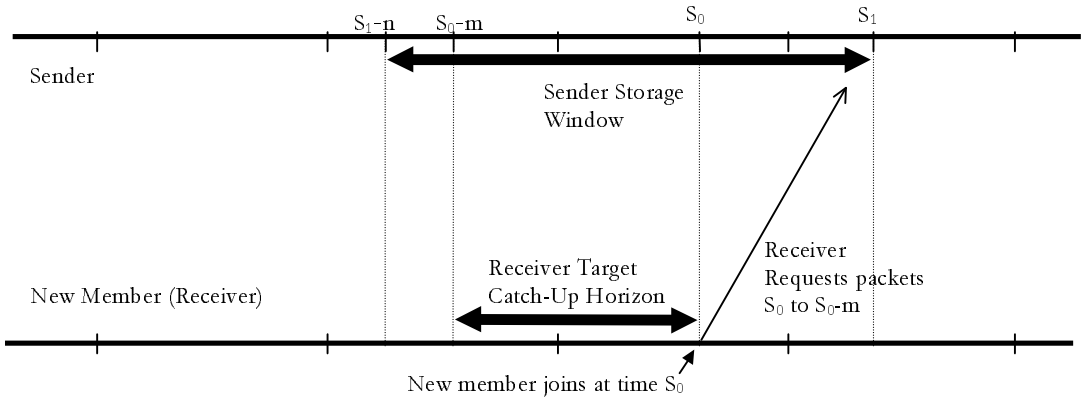


Figure 8 - (n,m) Persistence Windows

4.4. Persistence

In Section 3.1.5, we introduced and discussed three modes of *Persistence*: temporal, sequence number-based, and logical. TMP supports sequence number-based and logical persistence with two methods of specifying the lifetime of packets, *(n,m) persistence windows* and *logical persistence windows*. In both methods, we have coupled the notions of storing packets for error correction with the mechanisms required to bring new members up to the current state.

The (n,m) persistence window method supports sequence number-based persistence. Packets are retained or discarded strictly based on the sequence number of the packet. (n,m) persistence windows are specified by the following parameters:

- **n** - The number of packets stored by the sender. As shown in Figure 8, *n* specifies the size of the window for storing packets at each sender. For example, a packet with a sequence number of *s* is discarded once a packet is sent with a sequence number greater than *n* + *s*.
- **m** - The number of packets that new members will request and receive from each sender. We will refer to this value as the *target catch-up horizon* for new members. For example, if a new member receives a packet with sequence number *w*, the new station will request packets *w* - *m* to *w* -*1*. Packets prior to *w* - *m* will not be requested by new stations.

Clearly it must always be the case that $n \geq m$. If this were not the case, new members would attempt to retrieve packets already discarded by the source. In Figure 8, we show the case of a new receiver joining at sequence number S_0. Since the receiver's target catch-up horizon is *m*, the receiver requests retransmission of packets S_0 to S_0-*m*.[14] Figure 8 depicts the sender receiving this request when its sending window has advanced to S_1. Since S_1-$n < S_0$-*m*, the sender still has the packets within its storage window and is able to retransmit the packets. It is clear that *m* and *n* must be chosen such that error correction is possible despite the latencies associated with requests for retransmission and responses from senders. If *n* and *m* are too close together or not large enough, packets may be discarded before error correction can occur. We define several special cases of the parameters *n* and *m* to define boundary conditions:

(∞ , 0) - This defines the case in which senders store *all* transmitted packets since the beginning of the multicast session. However, new members do not request retransmission of any packets that were transmitted before they joined the group.

[14] Note that the sequence number space is defined strictly relative to the sender (implying single source ordering).

(∞ , ∞) - This defines the case in which senders store all packets transmitted since the start of the session and new members request and receive all packets transmitted as well. This is the service property defined by SRM [FLO95].

(0 , 0) - This defines an unreliable service. Clearly without storing packets, the sender is unable to retransmit packets to correct errors in transmission.

The other method of specifying session persistence that is introduced is *Logical Persistence Windows*. This method allows applications to define persistence not in terms of strict temporal or sequence number properties but by a logical partitioning of the data as defined by the application. In TMP, each data packet is contains a label called a *StreamID* that defines the logical partitions to which the data packet belongs. Each unique StreamID defines a Stream or logical partition. To guarantee that all StreamID's are unique, a StreamID consists of two parts: the SSRC of the station creating the Stream and a unique stream number. Since SSRC's are unique to each station and the sequence numbers are defined by that station, uniqueness is ensured. For example, the first stream created by the station with the SSRC of A is labeled (A,1). A stream can define pages created by participants in a shared whiteboard. StreamID's are based on the logical page identifiers referred to in [FLO95].

Logical persistence windows are defined by the following parameters:

- **sp** - The number of partitions (Streams) stored at each sender.
- **rp** - The number of partitions (Streams) that new members request from each sender when joining. For example, station A joins and receives a packet from station B with StreamID (B, 3). If *rp = 2*, station A will request streams (B,3) and B(,2). This value defines a logical "catch-up horizon" at each receiver similar to the sequence number-based catch-up horizon defined for (n,m) persistence windows.

The restrictions on values chosen for *sp* and *rp* are similar to that in the previous discussion of (n,m) persistence windows. It is clear that $sp \geq rp$ to prevent receivers requesting partitions of data that are no longer available. Generally, it is sufficient for $sp = rp - 1$ to ensure reliable delivery[15].

4.5. The TMP Application Program Interface (API)

The Tunable Multicast Protocol API presented in this thesis is specific to the Windows implementation but can be modified to support additional platforms. The API has been constructed so that it resembles the asynchronous Winsock API as much as possible[QUI95] [HAL93]. By providing an interface with which programmers are likely to be familiar, we hope to accelerate application development. Communication between the application and the protocol is asynchronous and non-blocking. The asynchronous communications model on the existence of a message loop such as that provided by the Windows API. The primary object in the TMP protocol is the MSocket. The application controls the protocol through exposed MSocket methods and through parameters passed to the MSocket constructor. The protocol communicates with the application through a single exposed method called MRead. The protocol notifies the application that data or control information is available by sending a Window Message to the application's main window. The application must respond to this message by executing MRead commands. We will discuss the MRead command in detail in the following sections. Throughout the TMP API, the return value of methods is often boolean. A value of false is used to indicate

[15] This assumes that logical partitions are not advanced so quickly that partitions are discarded more quickly than error correction can occur..

error conditions. This clearly should be replaced by integer return values indicating error codes.

Multicast sessions are created by declaring an object of type `MSocket`. The exposed interface to the MSocket object is shown in Figure 9. The MSocket constructor serves only to initialize some of the internal members of the MSocket class and takes no arguments. The other member functions will be discussed in detail. Both MAsyncSelect and CreateMCast must be called prior to use of the MSocket.

CreateMCast*:*

This mandatory method executes the primary initialization of the underlying multicast IP sockets and sets the values that will indicate the service provided. The method will return TRUE if the initialization was successful and FALSE otherwise. The parameters are as follows:

- nSocketPort - An unsigned integer indicating the port for the reliable multicast socket.
- *pAddress - A pointer to the character string containing the multicast IP address of the group to be joined. The string is in standard notation e.g. "224.228.22.5".
- Ordering_Mask - An unsigned integer indicating the ordering services desired for the session. The acceptable values are UNORDERED, SINGLE_ORDERED, and LOGICAL_ORDERED. The TMP header file includes the constants for these values. These values correspond to the unordered, single source ordering, and logical source ordering discussed in previous sections. Currently, this value is ignored by the protocol and defaults to logical ordering.

```
class MSocket {

      MSocket(UINT MaxThreadCount);

      ~MSocket();

      BOOL  CreateMCast (UINT nSocketPort, char *pAddress,
                         UINT Ordering_Mask, UINT Persistence_Mask,
                         int Receiver_Storage, int Sender_Window
                         int Catch_Up);

      BOOL  MAsyncSelect (HWND ParentHwnd, UINT UserMessage);

      BOOL  SetFlowControl (UINT TokenRate, UINT Max_Burst);

      unsigned long GetLocalSSRC();

      BOOL  SetActiveStream (unsigned long SSRC, unsigned long SeqNum);

      BOOL  GetActiveStream (unsigned long &SSRC, unsigned long &SeqNum);

      int   MWrite (char *pData, unsigned short &length,
                    unsigned long &app_type, StreamID &stream_id);

      int   MRead (char *pData, unsigned short &length,
                   unsigned long &app_type, StreamID &stream_id);

}
```

Figure 9 - MSocket Object Interface

- Persistence_Mask - An unsigned integer indicating the reliability service desired for the session. The acceptable values are LOGICAL, NMWINDOW, and RECEIVER_NONE. Again these values are defined as constants in the TMP header file. The RECEIVER_NONE can be logically OR'd with either of the two other values to indicate that throughout the entire session that *all* receivers will not store data or respond to requests for retransmission. The choice of LOGICAL or NMWINDOW indicates which model of persistence was chosen for the session (as discussed in Section 3.5). Currently, this value is ignored by the protocol and defaults to LOGICAL.

- Receiver_Storage - An integer representing either the number of packets or the number of logical streams stored for each sender (depending on the value of the Persistence_Mask). A value of 0 indicates that the given receiver will not store

packets.[16] Values less than 0 indicate total storage at each receiver. Currently, this value is ignored by the protocol and defaults to infinite storage.

- Sender_Window - An integer representing either the number of packet or the number of logical streams stored by senders (depending on the value of the Persistence_Mask). A value of 0 is only allowed with an unreliable service. Values less than 0 indicate total storage at each sender. This value corresponds to the *n* value described in Figure 9. Currently, this value is ignored by the protocol.

- Catch_Up - An integer representing either the maximum number of packets or the logical streams that the new member will attempt to "catch-up" from each existing member of the session. This value corresponds to the *m* value described in Figure 9. Currently, this value is ignored by the protocol.

Clearly, there are possible combinations of these parameters that are not appropriate. For example, if the value of Catch_Up exceeded the value specified for Sender_Window, new members would attempt to retrieve data that was no longer available from the original source of the data. In cases such as this, the function will return FALSE to indicate an error condition.

MAsyncSelect:

This required function provides the link between the MSocket object and the application's message loop. It provides similar functionality to the Winsock WSAAsyncSelect command [HAL93]. The parameters are as follows:

- ParentHwnd - A handle to a the window to which a message should be sent whenever data or control information is ready to be read. This is analogous to

[16] This allows individual stations to choose to play a more passive role in the group. Unless all receivers are guaranteed not to respond to repair requests other than those for which they are the source, the delay associated with response cannot be eliminated (hence the special mask value for persistence).

the FD_READ condition described in standard BSD Sockets programming [COR91].

- UserMessage - The Windows Message (an unsigned integer) that will be sent to the window specified in ParentHwnd.

The application can then read from the MSocket simply by executing an MRead command (discussed later in this section) in response to receipt of the UserMessage.

SetFlowControl:

This is the function used to set the flow control parameters for all data and retransmissions. This function is not considered mandatory but should be used immediately following the MAsyncSelect command prior to first use. Otherwise, default (restrictive) flow control will be provided. The parameters for this function correspond to those for standard "leaky bucket" flow control as follows:

- TokenRate - The number of bytes added to the "bucket" per millisecond. This value corresponds to the long term outgoing throughput. For example, a value of 200 will give a throughput of 200 kbytes/second.
- Max_Burst - The maximum number of bytes that can be transmitted within a negligible period of time measured in bytes. This corresponds to the size of the "bucket". A value of 5000 corresponds to a maximum allowable burst of 5 kbytes.

The value (0,0) is reserved and indicates a dynamically adjusted flow control mechanism.[17] This function can be executed by the application at anytime and can be used to manually adjust the protocol in response to error rates, congestion, etc.

[17] Although unimplemented an example algorithm can be found in [BUS96].

GetLocalSSRC:

The Synchronization Source Identifier (SSRC) is dynamically determined by the protocol. In order to enable the application to specify that SSRC when creating a new Stream, it is necessary that the application have a way to determine its own SSRC. The function takes no arguments and returns the SSRC of the local station. Error conditions are indicated by returning a value of 0.

SetActiveStream:

This function is used to identify the Active Stream. The Active Stream receives priority of retransmission and is the primary logical stream identified by the RTCP heartbeat report. Rather than specifying in the protocol that the Active Stream will match the most recent incoming or outgoing ADU, this function gives the application explicit control. The two parameters (SSRC and sequence number) completely specify a StreamID.

GetActiveStream:

This function is simply an accessor function to return the Active Stream. By allowing the protocol to store this information, the application can simply call this function when the Active Stream is needed rather than explicitly storing that information. The parameters are self explanatory.

MWrite:

This function is used to transmit new data to the multicast group. The parameters identify the Stream to which the data will be sent as well as the type of data. This function

is non-blocking by design. Since the protocol makes a copy of the data, the application is free to delete the memory used for the data immediately upon return from this function. The parameters are as follows:

- pData - A pointer to the character string containing the data to be transmitted.
- length - An unsigned short containing the length of the data. Data segments are limited to 256 bytes.
- app_type - A 32 bit type field whose meaning is defined by the application. Using this field, the application can define different packet types to be used as the application determines appropriate. There are a number of reserved types that will be discussed during the discussion of MRead.
- stream_id - The StreamID object identifying the logical partition to which the data belongs.

This function will return a value less than 0 in response to error conditions.

MRead:

This function is used by the protocol for two purposes: to present data from the multicast group to the application and to communicate events. These events are identified through the use of reserved values in the app_type field . Applications call this function in response to the *UserMessage* Windows Message that is specified in the MAsyncSelect function call. The parameters are as follows:

- pData - A pointer to the character string containing the received data. For the special event types, this field contains data associated with the event.
- length - An unsigned short containing the length of the data. Data segments are limited to 256 bytes.

- app_type - An 32 bit type field whose meaning is defined by the application. Using this field, the application can define different packet types to be used as the application determines appropriate. The special reserved types defined in the TMP header files. Their meanings are shown in Table 2:

Reserved Type	Data Field Contents	Description
NAK_DENY	Sequence number unavailable	Used to indicate the error condition that the ADU is no longer available from the group.
GROUP_JOIN	SSRC of new member	Sent to indicate a new member joining the group
GROUP_LEAVE	SSRC of member leaving	Sent to notify the application of a member leaving the group

Table 2 - Reserved Application Type Values

- stream_id - The StreamID object identifying the logical partition to which the data belongs.

The application then must respond in a meaningful way to both the application defined types as well as the reserved types that indicate certain events. Alternatively, applications unconcerned with an event notification may ignore the notification.

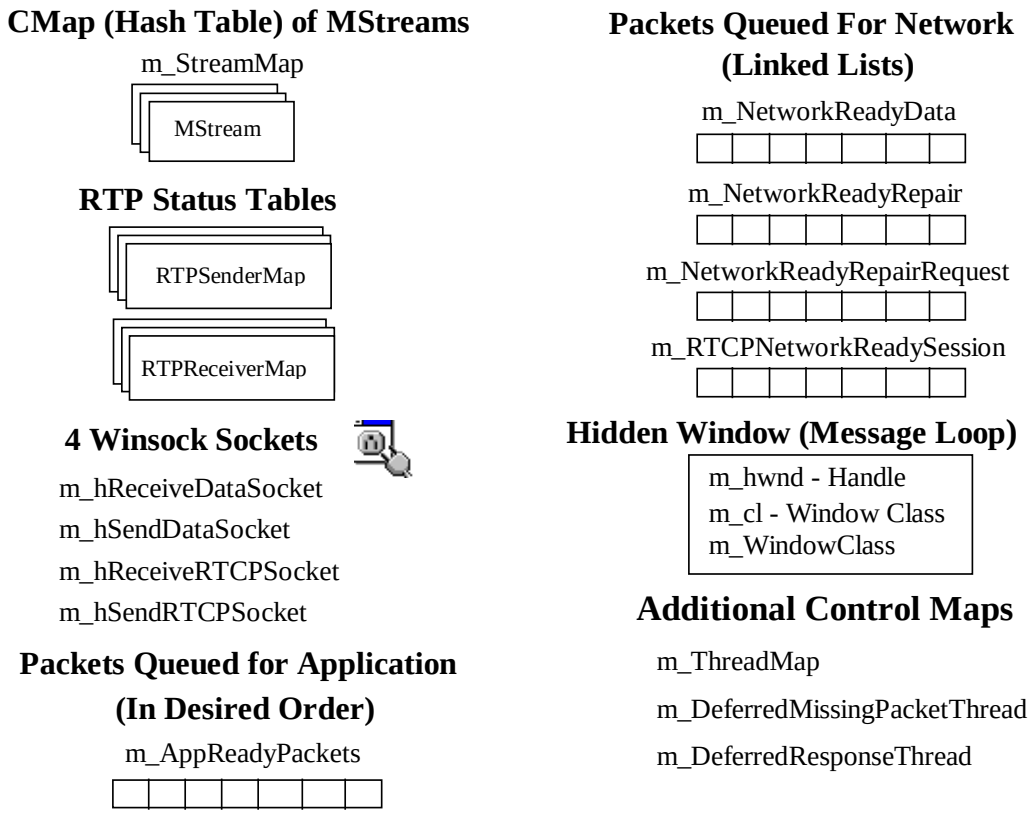


Figure 10 - MSocket Object Diagram - Major Components

4.6. Internal MSocket Object Model

The API presented in Section 4.5 only specifies the external interface to the TMP protocol. In this section, we present an overview of TMP's internal object structure. This overview is intended to cover the highlights of the object structure rather than as an exhaustive reference.

TMP's primary object is the MSocket. The MSocket is the only object that is visible to the application. The other main objects in TMP are the MStream and MWriter objects. These objects are discussed in Sections 4.6.2 and 4.6.3.

4.6.1. MSocket Object

Figure 10 depicts the major components of the MSocket object. Throughout the TMP implementation, we use an associative hash table array known as a CMap provided by the Microsoft™ Foundation Class (MFC) library. This object provides immediate access to objects contained in the CMap without iterating through the entire list. A drawback to this container class is that iteration through all elements in the CMap is computationally expensive. Furthermore, an initial size for the hash table is specified on construction of a CMap and the hash table must be resized if it grows beyond the initial size given.

The MStream objects stored in the m_StreamMap member variable are the primary objects contained in the MSocket. The object structure of the MStream is discussed in Section 0 and depicted in Figure 11. As we discussed in Section 4.5, each TMP packet contains a StreamID and the SSRC of the source of the packet. TMP processes packets in MStream objects based on the StreamID field. Each unique StreamID corresponds to a separate MStream object. When packets and control information is received from the network, it is passed to the appropriate MStream object for processing.

The RTP status maps store information about each known SSRC. This information includes all the information required to send RTCP Sender and Receiver Reports. As packets are received, they are first processed by the MSocket to capture the RTP information before being passed on to the appropriate MStream object for processing.

The MSocket object contains four multicast sockets to perform low-level sockets operations. Handles to these sockets are included member variables of the MSocket object. There are two pairs of sockets, one for RTP data and one for RTCP control

packets. The sockets are bound to successive ports on the same multicast address with the data sockets bound to an even port and the control sockets bound to the next consecutive odd port. As shown in Figure 10, we use separate sockets for sending and receiving to simplify the sockets code. Each of the sockets uses the Winsock asynchronous notification mode for communication and is configured in a non-blocking mode. When data available on the receiving sockets, a Windows Message is sent to the hidden window that is constructed as part of our MSocket object. The only reason that the hidden window is present is to serve as an event loop though which the low-level socket notification can occur.

We use linked lists to store packets that are either queued for transmission to the network or queued for delivery to the application. A single liked list contains all packets that are waiting to be read (through the MRead method) by the application. Packets are only placed in the "App Ready" list when the desired ordering guarantees are ensured. As shown in Figure 10, TMP maintains separate queues for the different packet types transmitted to the network. For efficiency, these queues consist of pointers to the actual packets so that additional copy operations are avoided. This has a number of advantages over a single queue implementation:

- Rate control can be applied separately to each queue more easily. The protocol can be easily adapted to apply different rate control mechanisms to each type of data. For example, this would allow different leaky bucket parameters for original transmissions and retransmissions.
- Separate queues enables the protocol to be tuned to place different priorities on different packet types. For example, an application might want to place a higher priority on error correction than on the transmission of new data.

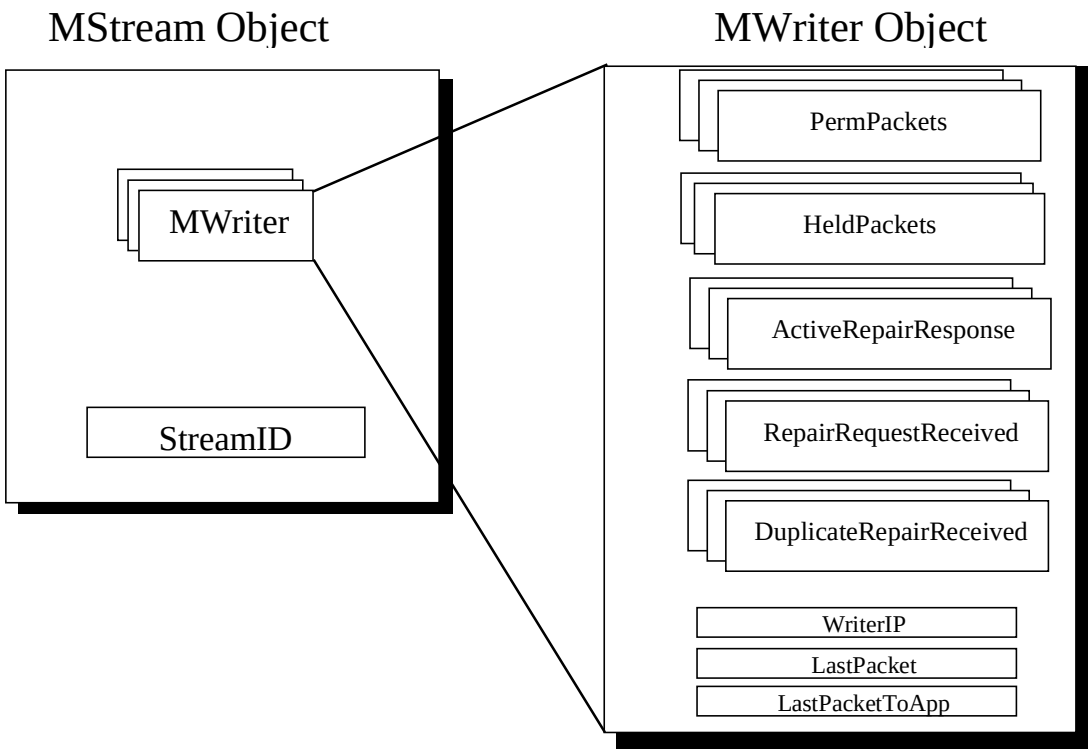


Figure 11 - MStream and MWriter Objects

The additional control maps depicted in Figure 10 are used to provide the thread limiting and control operations described in Section 4.2. Each time a thread is created, a handle to that thread is added to the ThreadMap so that the application can terminate all active threads prior to termination. The other two maps are used to queue threads (FCFS) for both requesting error correction and responding to other stations requests for error correction. These thread maps contain objects that consist of all the information needed to create the missing packet or repair response thread. For example, a missing packet thread object contains the packet numbers for which the thread will request and verify receipt. These queues prevent a large number of threads from degrading performance.

4.6.2. MStream Object

As discussed in Section 4.4, each MStream object corresponds to a logical partition of data. To understand the role of an MStream, it is important to note that multiple stations can write to the same Stream. For example, if station A creates a page with a StreamID of (A , 1) on a shared whiteboard, other stations can transmit data with that StreamID that "writes" to the page in the whiteboard that station A created. The MStream object merely serves as a collection of the more important object, the MWriter. The relationship between the MStream and MWriter object is depicted in Figure 11. MWriters are stored in the WriterMap and can be accessed by the unique SSRC field. MStream methods generally determine which MWriter object is affected by the operation through a lookup on a SSRC and then call the corresponding operation on the MWriter object.

4.6.3. MWriter Object

Figure 11 depicts the major components of an MWriter object. The MWriter is the only object in TMP that stores packets, determines persistence, and ensures reliability. The PermPackets CMap is the container for the packets that TMP stores for error correction. The other CMaps shown in the figure (HeldPackets etc.) contain only the sequence numbers of the packets concerned rather than the packets themselves. The HeldPackets CMap is used as a buffer containing sequence numbers so that packets can be delivered to the application in the proper order.

The error correction algorithms presented in Section 4.2, require that a significant amount of state information be stored in the MWriter object. The error correction

algorithm depicted in Figure 6 requires that the protocol determine if other stations have requested the same missing packets. This is accomplished by keeping track of repair requests in the RepairRequestReceived CMap. Similarly, the repair response algorithm depicted in Figure 7 requires that the protocol determine if other stations have already sent a retransmission or if the request is redundant. Duplicate repairs are avoided by tracking repairs in the DuplicateRepairReceived CMap and the MWriter tracks the ongoing repair response threads in the ActiveRepairResponse list to prevent the station from starting multiple threads for redundant requests.

5. Conclusions

With the rapid expansion of Internet bandwidth and multimedia technology, computer networks are becoming the means through which remote collaboration occurs. Reliably multicasting data to large groups of participants is the next step in the evolution of multicast tele-collaborative applications. At present, no reliable multicast protocol exists that can satisfy the demands of most tele-collaborative applications. Conventional wisdom says that it is not possible to find a single reliable multicast protocol that can meet the needs of all applications. In contrast, we submit that the needs of most tele-collaborative applications can be met through the TMP protocol introduced in this thesis.

Our approach to reliable multicasting is specifically tailored to meeting the needs of tele-collaborative applications without attempting to provide services that are not scalable to large group sizes. The emphasis that many sender-initiated reliable multicast protocols place on total reliability and total ordering has made them too inefficient to support large tele-collaborative groups [PIN94]. An existing protocol, called SRM, relaxes the strict ordering and reliability constraints of sender-initiated protocols but requires large amounts of memory to ensure reliability [FLO95] [LEV96]. By introducing new mechanisms that allow us to relax traditional notions of reliability, we have eliminated the extreme memory requirements possible with SRM without compromising reliability.

The TMP protocol, as presented here, has been used to develop a number of tele-collaborative applications including a multicast file distribution application and a shared

whiteboard[18]. The protocol is available as a Windows DLL that can be used to accelerate the development of many windows based applications. Informal tests have verified the NAK and response implosion algorithms. The maximum throughput of TMP hardware dependent but early results indicate throughput on the order of 256 kbytes/sec. on Pentium 100 class machines. Since the algorithms implemented in the TMP protocol only improve on the scalability claimed by the SRM protocol, we anticipate wide-scale testing to indicate a maximum group size supported to be equal to or greater than SRM.

We do not consider the protocol to be a completed work but rather as a base from which to continue research. We will discuss future work both for the TMP protocol as well as the general topic of reliable multicast in Section 5.2.

5.1. Summary of Contributions

The primary contribution of this thesis comes from a detailed exploration of the issues involved in providing a reliable multicast protocol for use in tele-collaborative applications. Through this exploration and research, we have made the following contributions:

- In Chapter 2, we survey existing reliable multicast protocols and highlight the characteristics of those protocols that are fundamentally incompatible with supporting large tele-collaborative groups.
- In Chapter 3, we present a new approach to reliable multicast specifically tailored to the demands of tele-collaborative applications. We reject the popular belief that a

[18] The shared whiteboard application was written in Java and communicated through a multicast reflector that uses TMP.

single protocol cannot efficiently support a wide range of applications in favor of a single tunable, reusable, and flexible protocol capable of supporting many tele-collaborative applications. We advocate a receiver-initiated approach to reliability in order to support larger groups efficiently. To achieve this level of flexibility and reuse, we have eliminated total ordering and rigorous maintenance of group membership in favor of less demanding service models.

- We present a model that allows us to represent the design space for reliable multicast protocols. In particular, we introduce two new dimensions in the design space called "Persistence" and "Receiver Storage". By exploiting the notions of persistence and receiver storage we are able to overcome shortcomings of previous receiver-initiated protocols.

- In Chapter 4, we present the Tunable Multicast Protocol (TMP). The design of this protocol comes directly from our systematic analysis of our Reliable Multicast Design Space. TMP implements the new models of Persistence and Receiver Storage introduced in the previous chapter and uses tuning parameters to provide flexible ordering and reliability services.

5.2. Future Work

This thesis presents a number of avenues for future research.

Clearly, the TMP protocol needs to be subjected to rigorous formal testing and analysis to verify the performance and reliability of the algorithms presented here. Extensive testing must be conducted to verify the performance of the persistence modes described in Section 4.4. In particular, it is important to verify the reliability of the TMP protocol for both logical and sequence number-based persistence windows.

One of the primary goals of TMP was to provide a reliable multicast protocol that accelerates the development of multicast applications. Towards this goal, the TMP protocol has been compiled into a Windows dynamic link library (DLL) that can be easily integrated into new applications without the need for low-level multicast sockets programming. We have already used this DLL to design a number of applications including a multicast file distribution application, a multicast shared "whiteboard" written in Java, and a multicast "Chat" application. The easy reuse of the TMP dynamic link library has greatly enhanced the development of these applications.

One future application of the TMP protocol is tele-conference control applications. As an example, The Grounds Wide Tele-tutoring System (GwTTS), developed at the University of Virginia in the Summer of 1995, provides a web-based multicast conference control service [LIE96]. The conference control protocol that was initially implemented used multicast to unreliably transmit session information to group members. However, wide-area demonstrations made it clear that reliability is necessary to ensure that all participants receive the same control information. The TMP protocol is ideally suited to replace the current unreliable conference control protocol in the Grounds Wide Tele-tutoring System. The modular, reusable design of the protocol makes it simple to integrate TMP into the current program structure and effectively shield the GwTTS developers from the complexity of the reliability algorithms.

Another area for future research is in the integration of a probabilistic algorithm for NAK suppression into the TMP protocol. As described in Section 4.2.1, the current NAK suppression algorithm has all stations delay for a random time period before requesting retransmission. Since the RTCP protocol provides each receiver a "best guess"

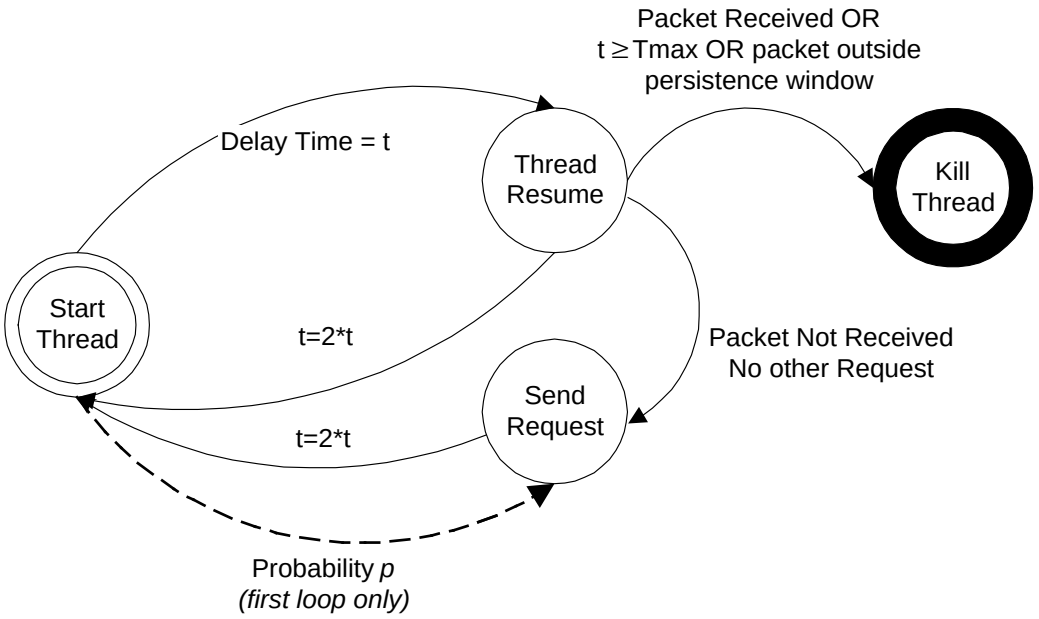


Figure 12 - Probabilistic Error Correction Thread Algorithm

approximation of the size of the multicast group, it is possible for stations to request retransmission immediately, based on a probabilistic estimate of the number of stations that experience a similar error. The NAK suppression algorithm can be modified so that a station detecting a loss sends a request immediately with probability *p,* (shown in Figure 12) where

$$p = \frac{\text{Estimated \#Stations with Error}}{\text{Estimated Group Size}}$$

This probabilistic approach to NAK suppression may reduce the time required to correct errors.

Reliable multicasting is a rich and dynamic field of research that has only recently begun to mature. As multicast routing protocols and infrastructure improve, the ultimate goal of reliably multicasting data to thousands and even millions of receivers is not beyond

reach. The tele-collaborative applications mentioned in this thesis only scratch the surface of the possibilities and benefits that may be realized through the use of efficient reliable multicast protocols. However, great network protocols are of little use unless they can be effectively used by application developers. TMP emphasizes the importance of stressing software engineering as reliable multicast protocols continue to develop.

BIBLIOGRAPHY

[ARM92] S. Armstrong, A. Freier,A and K. Marzullo. *"Multicast Transport Protocol"*, RFC 1301, Internet Engineering Task Force, February 1992.

[BIR87] K. Birman and T. Joseph. "Reliable Communication in the Presence of Failures". ACM Trans. Comp. Syst., 5(1):47-76, February 1987.

[BIR88] K. Birman, T.A. Joseph, K. Kane. *"ISIS - A Distributed Programming Environment User's Guide and Reference Manual, First edition.* Dept. Of Computer Science, Cornell University March 1988.

[BIR91] K. Birman, A. Schiper, P. Stephenson, *"Lightweight Causal and Atomic Group Multicast"*, *ACM Transaction on Computer Systems.* 9(3): 272-314. August 1991.

[BOR94] C. Bormann, J. Ott, H. Gehrcke, T. Kerschat, and N. Seifert. *"MTP-2: Towards achieving the S.E.R.O. Properties for Multicast Transport".* In ICCN '94, San Francisco, California, 1994.

[BUS96] I. Busse, B. Deffner, and H. Schulzrinne, *"Dynamic QoS control of multimedia applications based on RTP,"* Computer Communications, Jan. 1996.

[CAS94] S. Casner. *"Are you on the MBone?". IEEE Multimedia.* Vol. 1, No. 2. Summer 1994.

[CHA84] J. M. Chang and N. F. Maxemchuck, *"Reliable Broadcast Protocols", ACM Transactions on Computing Systems*, 2(3):251-273. August 1984.

[CHT96] T. Chandra, V. Hadzilacos, S. Toueg, and B. Charron-Bost, *"On the Impossibility of Group Membership"*, Cornell University, 1996.

[CLA90] D. Clark and D. Tennenhouse. *"Architectural Considerations for a New Generation of Protocols", Proceedings of ACM SIGCOMM '90*, pp. 201-208. Sept. 1990.

[COH96] J. Cohoon and J. Davidson. *"C++ Program Design, An introduction to Programming and Object-Oriented Design".* Irwin Publishing. 1996.

[COR91] Comer, D., *"Internetworking with TCP/IP Volume I: Principles, Protocols, and Architecture"*, Prentice Hall, Englewood Cliffs, New Jersey. 1991.

[DAN89] P. Danzig. *"Finite Buffers and Fast Multicast", ACM Sigmetrics Performance Evaluation Review '89.* Vol 17. pp. 108-117. May 1989.

[DEE89] S. Deering, *"Host Extensions for IP Multicasting".* RFC 1112. Internet Engineering Task Force. August 1989.

[DEE91] S. Deering. *"Multicast Routing in a Datagram Internetwork".* PhD thesis, Stanford University, Palo Alto California, December 1991.

[DEE95] S. Deering, D. Estring, D. Farinacci, V. Jacobson, C.G. Liu, L. Wei, *"Protocol Independent Multicast(PIM): Sparse Mode Protocol Specification".* Internet Draft March 1994.

[DEM94] B. Dempsey, M. Lucas, and A. Weaver. *"Design and Implementation of a High-Quality Video Distribution System using XTP Reliable Multicast". Multimedia: Advanced Teleservices*

and High-Speed Communications Architectures, Ralf Steinmetz (Editor), Spring-Verlag, pp. 376-387, Heidelberg, Germany, Sept. 1994.

[DIO96] C. Diot, W. Dabbous, and J. Crowcroft. *"Multipoint Communication: A Survey of Protocols, Functions and Mechanisms"*, INRIA and University College of London, September 1996.

[ELM94] R. Elmasri and S. Navathe. *"Fundamentals of Database Design, 2nd Edition"*. Benjamin/Cummings Publishing Company, Redwood City, CA. 1994.

[FLO95] S. Floyd, V. Jacobson, S. McCanne, C.G. Liu, and L. Zhang. *"A Reliable Framework for Light-Weight Sessions and Application Level Framing"*, *ACM SIGCOMM '95*, Boston. August 30-September 1, 1995.

[FRE] R. Frederick, Nv Manual Pages.

[FRMA90] A. Frier, K. Marzullo, *"MTP: An Atomic Multicast Transport Protocol"*, Cornell University, 1990.

[GAR91] H. Garcia-Molina and A. Spauster. *"Ordered and Reliable Multicast Communication"*. *ACM Transactions on Computer Systems,* 9(3):242-271, August 1991.

[HAL93] M. Hall et al *"Windows Sockets An Open Interface for Network Programming under Microsoft® Windows™ Version 1.1"*, January 1993.

[HIL95] M. Hiltunen and R. Schlichting. *"Properties of Membership Services"*. In *Proceedings of the 2nd International Symposium on Autonomous Decentralized Systems*. Phoenix Arizona. April 1995.

[HOL95] H. Holbrook, S. Singhal, and D. Cheriton. *"Log-based Receiver-reliable Multicast for Distributed Interactive Simulation"*. In *Proceedings of ACM SIGCOMM'95*, pp 126-135. 1995.

[JAC92] V. Jacobson and Steve McCanne. *"The LBL Audio Tool VAT"*. Manual Page. July 1992.

[JON91] M.G.W. Jones, S.-A Sorensen, and S. Wilbur. *"Protocol Design for Large Group Multicasting: The Message Distribution Protocol", Computer Communications,* 14(5):287-297, 1991.

[KAA89] Kaashoek M.F.,Tanenbaum A.S., Flynn Hummel S., Bal H.E. *"An Efficient Reliable Broadcast Protocol", Operating Systems Review* Vol. 23, No. 4, pp 5-20. (October 1989)

[KAA91] Kaashoek M.F.,Tanenbaum A.S.. *"Group Communication in the Amoeba Distributed Operating System"*. In *Proceedings of the 11th International Conference on Distributed Computer Systems*, pp. 222-230. May 1991.

[LAM78] L. Lamport, *"Time, Clocks, and the Ordering of Events in a Distributed System"*. *Commun. ACM*. Vol. 21, N.7, pp. 558-565. July 1978.

[LEV96] B. Levine. *"A Comparison of Known Classes of Reliable Multicast Protocols"*, Masters Thesis, University of California Santa Cruz, June 1996.

[LIE95] J. Liebeherr, *"Multimedia Networks: Issues and Challenges"*, IEEE Computer, Vol. 28, No. 4, pp. 68-69, April 1995.

[LIE96] J. Liebeherr, et al., "The Grounds-Wide Tele-Tutoring System", Information available at http://www.cs.virginia.edu/~gwtts/.

[LIN96] J. Lin. and S. Paul *"RMTP: A Reliable Multicast Transport Protocol", Proceedings of IEEE Infocom '96*, March 1996.

[LUC95] M. Lucas. *"Survey of Multicast Tranport Ordering Protocols"*. University of Virginia. Unpublished 1995.

[LUC96] M. Lucas. *"Efficient, Low Latency Error Recovery and Receiver Feedback for Large Scale Multicast in Heterogeneous Networks"*. University of Virginia, PhD Proposal, February 1996.

[MAC94] M. Macedonia and D. Brutzman. *"MBone Provides Audio and Video Across the Internet"*, *IEEE Computer,* pp 30-36. April 1994.

[MEL94] P. Melliar-Smith, L. Moser, and V. Agrawala. *"Processor Membership in Asynchronous Distributed Systems. IEEE Transactions on Parallel and Distributed Systems,* 5(5):459-473. May 1994.

[MIL92] Mills, D., *"Network Time Protocol Version 3"*, RFC 1305, UDEL, March 1992.

[MIS91] S. Mishra, L. Peterson, and R. Schlichting. *"A Membership Protocol based on Partial Order"*. In *Proceedings of the IEEE International Working Conference on Dependable Computing for Critical Applications*. pp. 137-145 February 1991.

[MON94] T. Montgomery. *"Design, Implementation, and Verification of the Reliable Multicast Protocol"*, Masters Thesis, University of West Virginia. 1994.

[MOY94] J. Moy. *"Multicast Extensions to OSPF"*. RFC 1584, March 1994.

[PET89] L. Peterson, N. Bucholz, and R. Schlichting. *"Preserving and Using Context Information in Interprocess Communication". ACM Transactions on Computer Systems*, 7(3):217-246, August 1989.

[PIN94] S. Pingali, D. Towsley, and J. Kurose. *"A Comparison of Sender-Initiated and Receiver-Reliable Multicast Protocols", Proc. 1994 ACM Sigmetrics and Performance '94*, pp 221-230, May 1994.

[POS81] J. Postel, ed., *"Transmission Control Protocol – DARPA Internet Program Protocol Specification"*, RFC 793, USC/Information Sciences Institute, September 1981.

[QUI95] Quinn, B. and Shute, D., *"Windows Sockets Network Programming"*, ISBN: 0-201-63372-8, Addison-Wesley Publishing Company, Reading, Massachusetts. 1995.

[RAJ92] B. Rajagopalan, *"Reliability and Scaling Issues in Multicast Communication"*, *Communic. ACM,* pp. 188-198, (August 1992).

[RAM87] S. Ramakrishnan and B. Jain. *"A Negative Acknowledgement with Periodic Polling Protocol for Multicast over LAN". IEEE Infocomm '87.* pp. 502-511. March 1987.

[RIC91] A. Ricciardi and K. Birman. *"Using Process Groups to Implement Failure Detection in Asynchronous Environments"*. In *Proceedings of the 10th Annual ACM Symposium on Principles of Distributed Computing,* pp. 341-352. 1991

[SCH96] H. Schulzrinne, S. Casner, R. Fredrick, and V. Jacobson. *"RTP: A Transport Protocol for Real-Time Applications"*, RFC 1889, IETF, January 1996.

[SEG82] A. Segall and B. Awerbuch, *"A Reliable Broadcast Protocol"*, *IEEE Transactions on Communications*, 31(7) pp. 896-901, July 1983.

[STR92] W. Strayer, B. Dempsey, and A. Weaver. *"XTP: The Xpress Transfer Protocol"*, Addison-Wesley Publishing, July 1992.

[THY95] A. Thyagarajan and S. Deering. *"Hierarchical Distance-Vector Multicast Routing for the MBone"* ACM SIGCOMM 95'. Boston, MA. August 1995.

[TUR93] T. Turletti. *"H.261 Software Codec for Videoconferencing over the Internet"*. Technical Report 1834, Institut National de Recherche en Informatique et en Automatique (INRIA), January 1993.

[WAI88] D. Waitzman, C. Partridge, and S. Deering, *"Distance Vector Multicast Routing Protocol"*. RFC 1075. Internet Engineering Task Force. November 1988.

[WEA95] A. C. Weaver. *"Xpress Transport Protocol, Version 4.0"*. Available at http://www.cs.virginia.edu/~netlab/xtp_stuff/xtp4_tutorial.ps.

[WHE94] B. Whetten, T. Montgomerey, and S. Kaplan. *"A High Performance Totally Ordered Multicast Protocol"*. August 1994. Research Memorandum Available at: http://research.ivv.nasa.gov/projects/RMP/RMP.html

[YAV95] R. Yavatkar, J. Friffioen, and M. Sudan. *"A Reliable Dissemination Protocol for Interactive Collaborative Applications"*. *ACM Multimedia 1995*, pp 333-343. November 1995.

[ZAB96] Zabele, S, DeCleene, B, and Koifman, A, *"Reliable Multicast for Internet Applications"*, Proceeding of the Technical Conference on Telecommunications R&D in Massachusetts, March 1996.