EFFICIENT ERROR RECOVERY FOR RELIABLE MULTICAST

A Dissertation

Presented to the Faculty of the Graduate School

of Cornell University

in Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy

by

Zhen Xiao

January 2001

EFFICIENT ERROR RECOVERY FOR RELIABLE MULTICAST

Zhen Xiao, Ph.D.

Cornell University 2001

Multicast is an efficient mechanism for distributing data from one sender to multiple receivers. Many applications need a reliable multicast service which is not provided by the existing IP multicast protocol. Providing such a service on a large scale requires efficient algorithms for error recovery.

This dissertation presents a randomized reliable multicast protocol called RRMP which has demonstrably achieved several good properties. The protocol eliminates message implosion by diffusing the responsibility of error recovery among all members in the group and improves the robustness of the system against process failures. It provides good local recovery by dynamically organizing members into an error recovery hierarchy according to their geographic locations. It optimizes buffer management through an innovative two-phase buffering algorithm that explicitly addresses the variances in delivery latency for large multicast groups. The algorithm reduces buffer requirements by adaptively allocating buffer space to messages most needed in the system and by spreading the load of buffering among all members in the group. The key idea of RRMP is to use randomization as a powerful technique to achieve high robustness and efficiency in reliable multicast communications.

The RRMP protocol works well within the existing IP multicast framework and does not require additional support from network routers. Both analysis and experimental results show that the performance penalty due to randomization is low and can be tuned according to application requirements.

BIOGRAPHICAL SKETCH

Zhen Xiao was born in Beijing, People's Republic of China. He received his B.S. in Computer Science from the Peking University in July 1996. In the fall of that year he entered the Ph.D. program in Computer Science at Cornell University. He was awarded an M.S. in May 1999 and graduated with a Ph.D. in January 2001.

To my parents

ACKNOWLEDGMENTS

I am grateful to my advisor, Kenneth P. Birman, for his guidance, encouragement, and support during my entire graduate study. He introduced me into the area of reliable multicast and provided many valuable comments on my research. Joseph Y. Halpern and Zygmunt J. Haas are the other two members in my committee. I want to thank them for their efforts and kindness.

I worked with Robbert van Renesse on various problems and wish to express my gratitude to him for many insightful discussions. Mark Hayden worked with me during my early years at Cornell University, from whom I gained valuable experiences on how to do research. I want to thank him for his knowledge and patience. Ken Birman and Mark Hayden are among the co-authors of the Bimodal Multicast protocol described in Chapter 3 of the dissertation.

This research was supported in part by ARPA/RADC under grant number F30602-99-1-0532 and NSF under grant number EIA 97-03470. Any opinions, findings, or recommendations presented in this dissertation are my own and do not reflect the official views of the funding agencies.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction

Multicast is an efficient mechanism for distributing data from one sender to multiple receivers. Instead of sending a separate copy of a message to each individual receiver, the sender sends only one copy of the message to a multicast group. Any receiver interested in receiving the data can join the multicast group. The IP multicast protocol [DC90] dynamically establishes a multicast tree from the sender to the set of receivers. The delivery service provided by IP multicast is unreliable: there is no guarantee that a multicast message can reach all receivers in a group. In addition, neither the sender nor any receiver has membership knowledge about the multicast group. Many applications, however, need a reliable multicast service. Examples include distributed system management, collaborative computing, distribution of software, and distributed interactive simulation (DIS). Providing such a service on a large scale requires efficient algorithms for error recovery.

One challenge in the design of an efficient error-recovery algorithm is *implosion avoidance.* As with any reliable multicast protocol, some members need to take responsibility for detecting message loss and performing retransmission. Putting this responsibility entirely on the sender leads to the well-known *implosion* problem: the storm of *acks* or *nacks* from a large set of receivers can overflow the sender's buffer and congest the network near it. Consequently, some reliable multicast protocols adopt a receiver-based reliability model in which a receiver is responsible for its correct reception of data. A well-known example is the Scalable Reliable Multicast protocol (SRM) [FJM$^+$95]. In SRM, when a member detects a message loss, it multicasts a retransmission request to the group. Any member holding a copy of the message can multicast a reply. A randomized back-off algorithm is employed to suppress duplicate requests and replies. More recently, Forward Error Correction (FEC) has been proposed in several reliable multicast protocols as an efficient technique for providing error recovery of uncorrelated loss in large multicast groups [McA90, Riz97, NBT98]. In these protocols, the sender takes k packets from the application and generates n encoded packets in such a way that any subset of k encoded packets are sufficient to reconstruct the original packets. The advantage of this approach is that a receiver can recover from up to $n - k$ packet losses without the need to ask for retransmissions.

Another challenge in the design of error control algorithms is how to confine the impact of a message loss to the region where the loss has occurred. This is especially important when the group is large. Experimental data collected over the MBone show that a large percentage of packets are lost by at least one receiver in large multicast groups [YKT96, Han97]. If retransmission requests and replies are multicast to the entire group, the network will be flooded with error control messages and a receiver will receive multiple copies of the same message. The original SRM protocol provides no local recovery, although recently some proposals have been made to localize the scope of error recovery [LESZ98].

For multicast applications with only one sender, several tree-based protocols have been proposed as an efficient way to avoid message implosion and to provide good local recovery. Examples include the Reliable Multicast Transport Protocol (RMTP [PSLB97]), the Log-Based Receiver-Reliable Multicast Protocol (LBRRM [HSC95]), and the Tree-based Multicast Transport Protocol (TMTP [YGS95]). In these protocols, receivers are grouped into local regions based on their geographic proximity. A repair server is selected in each region and made responsible for performing error recovery for all receivers in that region. Because a message loss can be repaired by a regional repair server rather than by the sender, this scheme reduces recovery latency and avoids message implosion at the sender.

If a receiver wants to perform error recovery for other receivers, it needs to buffer received messages for a certain period of time. Designing an efficient buffer management algorithm is challenging in large multicast groups where no member has complete group membership information and the delivery latency to different members could differ by orders of magnitude. Buffer management strategies vary widely in existing multicast protocols. The SRM protocol provides no buffering at the transport level. Instead it relies on the application to regenerate messages if necessary based on the concept of Application Level Framing [CT90]. In some tree-based protocols like RMTP, a repair server buffers all messages it has received in the current multicast session on a secondary storage. This allows a receiver to dynamically join a multicast session and still receive all messages sent in the session. However, for long-lived sessions, the amount of buffering at a repair server could become impractically large.

This dissertation presents our work of developing a randomized reliable multicast protocol called RRMP in an effort to address these challenges. The RRMP protocol has demonstrably achieved several good properties. It eliminates message implosion by distributing the responsibility of error recovery among all receivers in the group. It provides good local recovery by dynamically organizing receivers into an error recovery hierarchy according to their geographic locations. It optimizes buffer management through an innovative two-phase buffering algorithm that explicitly addresses the variances in delivery latency for large multicast groups. The algorithm reduces buffer requirements by adaptively allocating buffer space to messages most needed in the system and by spreading the load of buffering among all members in the group. The key idea of RRMP is to use randomization as a powerful technique to achieve high robustness and efficiency in reliable multicast communications.

The structure of the dissertation is as follows. We start with an overview of reliable multicast protocols in Chapter 2. We study previous work on virtual synchrony protocols, the scalable reliable multicast protocol (SRM), tree-based reliable multicast protocols, and router-assisted reliable multicast protocols. The purpose of the study is to identify a set of problems in existing error recovery algorithms and to provide background for the remainder of the dissertation. For example, virtual synchrony protocols provide strong reliability properties desired in settings such as stock exchanges, air traffic control systems, hospital settings, and other critical applications. However, providing such properties entails costly communications and limits the scalability of the protocol in large systems. This motivates our work on the Bimodal Multicast protocol described in Chapter 3. Bimodal Multicast employs a gossip-based anti-entropy algorithm for error recovery. Compared with traditional virtual synchrony protocols, it achieves a weaker reliability model but has better throughput stability. One problem with the Bimodal Multicast protocol, however, is that it makes no use of network topology information. Consequently, the protocol suffers from a tendency to do error recovery over potentially high latency links in the network, which limits its scalability in wide area settings. This motivates the design of the RRMP protocol which is the main focus of the dissertation. Chapter 4 describes the error recovery algorithm in RRMP which combines randomized error recovery in the Bimodal Multicast protocol and hierarchical error recovery similar to that employed by tree-based protocols. Formation of the error recovery hierarchy is described in Chapter 5. Chapter 6 describes the buffer management scheme in RRMP, which adopts an innovative two-phase buffering algorithm to explicitly address variations in delivery latency that arise in a wide area network. Chapter 7 evaluates the performance of the RRMP protocol using simulation. The results demonstrate that RRMP achieves fast error recovery with low overhead, compared with tree-based protocols. Chapter 8 presents experimental results of the protocol on the UNIX platform. Chapter 9 concludes the dissertation and points out several future research directions.

Chapter 2

Related Work

This chapter gives an overview of related work on reliable multicast protocols. It proceeds as follows. Section 2.1 describes virtual synchrony protocols. Section 2.2 describes the scalable reliable multicast protocol. Several examples of tree-based protocols are described in Section 2.3. Finally, Section 2.4 discusses router assistance for reliable multicast.

2.1 Virtual synchrony protocols

The *virtual synchrony* model was originally proposed in the ISIS project [BvR94, Bir93, BJ87a, BJ87b]. ISIS supports a *process group* abstraction in which each process is provided with a list of current members in the group. This is called a *view* of the group. The basic idea of virtual synchrony is to create an illusion to an application that distributed events are performed instantaneously and in lock-step, even though the underlying communication primitives only synchronize events to the degree that the application is sensitive to event ordering. To achieve this goal, group membership changes (i.e. view changes) are ordered relative to ongoing multicasts. Between two consecutive views, any two members which participated in both views always receive the same set of messages. The group communication model in ISIS has proven successful in several critical applications such as New York stock exchange and French air traffic control systems. It provides reliability guarantees that are strictly stronger than those provided by the remaining protocols described in this chapter.

Horus [vRBM96] is a successor of ISIS that provides a flexible architecture for protocol composition. In Horus, high-level communication protocols are implemented using a set of microprotocols. Each microprotocol has standard top and bottom interfaces and can be stacked on top of each other like Lego blocks. The benefit of this architecture is to allow an application to select a set of protocols to match its specific requirements without having to pay overhead for properties that it does not need. Figure 2.1 shows an example of protocol composition in Horus.

Ensemble [Hay98] is a successor of Horus and is written in a functional programming language called OCAML [Ler00]. Similar to Horus, Ensemble is extensively layered and highly reconfigurable. In addition, the use of functional programming language makes the system amenable to formal verification and optimization. The Bimodal Multicast protocol described in Chapter 3 was originally implemented as a protocol layer in Ensemble. The author has participated in the development of the Ensemble system.

Examples of other group communication systems include Transis [ADKM92], Totem [AMMS$^+$95], and Rampart [Rei94].

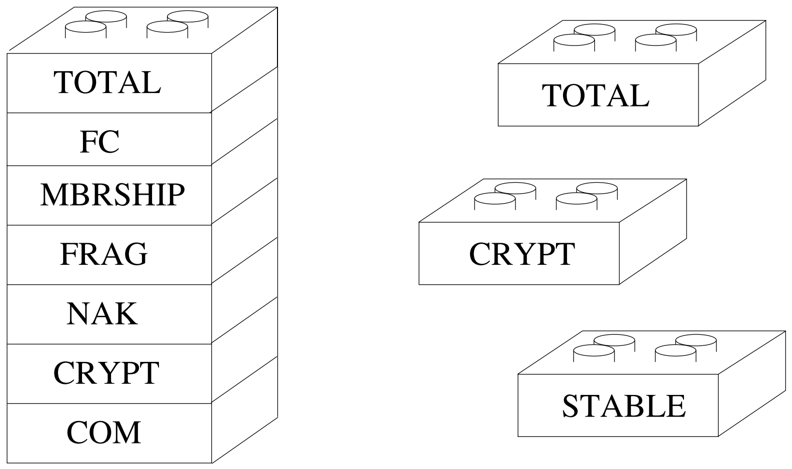


Figure 2.1: Protocol composition in Horus. Each microprotocol has standard top and bottom interfaces and can be stacked on top of each other like Lego blocks.

2.2 Scalable Reliable Multicast Protocol

The Scalable Reliable Multicast Protocol (SRM) adopts a receiver-based reliability model in which a receiver is responsible for its correct reception of data [FJM+95]. Reliability of this protocol is provided through collaboration of all receivers in the multicast group. SRM achieves good fault tolerance by multicasting retransmission requests and responses to the whole group. A randomized back-off algorithm is employed to suppress duplicate requests and replies. In this protocol, when a receiver detects a message loss, it waits a random amount of time before sending a request. The random delay period depends on its distance from the sender. The closer a receiver is from the sender, the more likely it will multicast a retransmission request. Hence if several receivers in the group all missed the same message, a receiver close to the point of message loss is likely to multicast the request first. If a receiver receives a request for the same message while it is waiting, it performs an exponential back-off and resets its request timer. Any member holding a copy of the message can multicast a repair. In order to suppress duplicate repair messages, a receiver waits a random amount of time before sending a repair. The random delay period is determined by its distance from the sender of the request message. If it receives a repair for the same message while it is waiting, it cancels its own repair. A receiver measures its distance from other receivers through periodic exchange of session messages among all receivers in the group. Session messages are also used to detect the loss of the last message in a burst.

In the ideal case, a message loss will trigger a single request immediately downstream of the point of message loss in the underlying multicast tree and a single repair immediately upstream of the point of loss. However, due to the randomized nature of the algorithm, it is possible for a message loss to trigger multiple requests and repairs, leading to redundant error control traffic.

Several researchers have discovered performance problems with the SRM protocol [BHO$^+$99, PPV98, PSLB97]. One of them is so-called the *crying baby* problem. In SRM, retransmission requests and replies are always multicast to the entire group. Hence if some receiver in the group has a much higher loss rate than others (such as a receiver behind a slow modem link), it will cause the network to be flooded with error control messages. A correct receiver may receive multiple copies of the same message. Recently some proposals have been made to localize the scope of error recovery in SRM [LESZ98, SEFZ98]. Another problem with the SRM protocol is that its randomized back-off algorithm, although helpful in avoiding message implosion, may increase error recovery latency.

2.3 Tree-based Reliable Multicast Protocols

For multicast applications with only one sender, several tree-based protocols have been proposed as an efficient way to avoid message implosion and to provide good local recovery. In these protocols, receivers in the group are organized into an error recovery tree based on their geographic proximity. A receiver missing a message contacts its parent in the tree for retransmission. Protocols in this category differ in how the error recovery tree is formed and how retransmission requests and repairs are sent. In the following we describe some best-known examples of tree-based protocols: RMTP [PSLB97], LBRRM [HSC95], TMTP [YGS95], and LGC [Hof97].

2.3.1 Reliable Multicast Transport Protocol

The Reliable Multicast Transport Protocol (RMTP) was originally designed for bulk data transfer [PSLB97], such as file transfer from a central server to a set of repositories. In this protocol, receivers are organized into local regions. In each local region, a Designated Receiver is selected to process acknowledgments from receivers in its region and to retransmit lost messages. When a receiver joins the group, it selects one of the Designated Receivers as its ACK Processor. Figure 2.2 shows an example where the multicast group is divided into three local regions. Each region has a Designated Receiver. When a Designated Receiver receives an acknowledgment from a receiver in its region, it identifies the set of messages that are lost. It is possible that several receivers missed the same message. For better efficiency, a Designated Receiver retransmits a message in unicast if the number of requests it received is smaller than some threshold. Otherwise it sends the message in multicast. Because a message loss can be recovered locally by a Designated Receiver rather than by the sender, this scheme reduces recovery latency and avoids message implosion at the sender. If a Designated Receiver itself misses a message, it gets a retransmission from its upstream Designated Receiver. RMTP allows a receiver to dynamically join a multicast session and still receive all messages sent in the session. To achieve this goal, each Designated Receiver buffers the entire set of messages in a secondary storage.

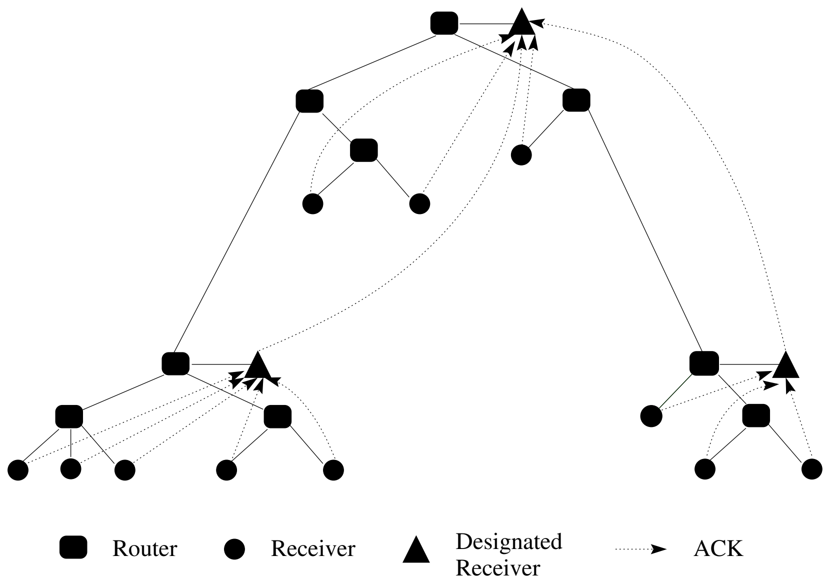


Figure 2.2: Error recovery hierarchy in RMTP. The multicast group is divided into three local regions with a Designated Receiver in each region.

The error recovery hierarchy in RMTP is static. It assumes that there is some application-level information about the geographic locations of receivers in the group. Based on this information, specific machines are chosen as Designated Receivers and are statically organized into a tree. Each Designated Receiver announces its presence by sending a SEND_ACK_TOME message down the multicast tree periodically. The

TTL field of the message is set to a pre-determined value. When a receiver receives SEND_ACK_TOME messages from several Designated Receivers, it chooses the most upstream Designated Receiver (in terms of hop counts) as its ACK Processor. Robustness of the protocol is achieved through soft-state timers. For example, if a receiver has not received SEND_ACK_TOME message from its Designated Receiver for some period of time, it may conclude that the Designated Receiver has failed. In this case, it switches to the Designated Receiver least upstream from the failed Designated Receiver as its new ACK Processor. Figure 2.3 shows a situation where one of the Designated Receivers, r, crashes. All receivers in r's region now send acknowledgments to an upstream Designated Receiver p. This, however, may lead to certain performance problems because message loss is no longer recovered locally. We will talk more about performance problems in tree-based protocols in Chapter 4.

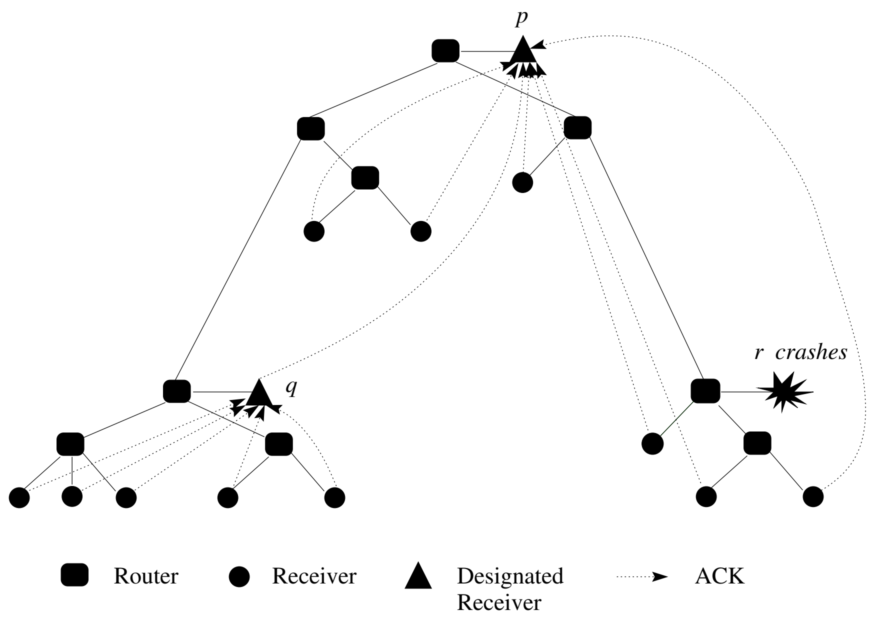


Figure 2.3: Failure of a Designated Receiver. Receivers in r's region now switch to an upstream Designated Receiver p.

2.3.2 Log-Based Receiver-Reliable Multicast Protocol

The Log-Based Receiver-Reliable Multicast Protocol (LBRRM) was designed for distributed interactive simulation [HSC95]. Applications in this category have relatively low update rates. However, once an update has occurred, receivers need to be notified of the update within short delay, even if the network may be lossy. This requires an efficient algorithm to detect a message loss and recover from it. To achieve this goal, each sender periodically multicasts a heartbeat message that includes the largest sequence number it has transmitted. In order to reduce the amount of network bandwidth consumed by heartbeat messages, LBRRM proposes a variable rate heartbeat scheme in which heartbeat messages are transmitted more frequently immediately after a data message. More specifically, when a sender first transmits a message, it initializes the interval between two heartbeat messages to a small value $h_{\min}$. This interval is doubled every time a heartbeat message is sent until it reaches a maximum limit $h_{\max}$. When the sender transmits another data message, it immediately resets the interval to $h_{\min}$. Compared with a fixed heartbeat scheme in which heartbeat messages are distributed evenly across time, this technique has low overhead while still providing fast loss detection.

Reliability of the protocol is achieved through a two-level hierarchy of logging servers: a primary logging server and a set of secondary logging servers. Receivers in the group are organized into *site*s based on their geographic locations. A site is a localized part of the network, similar to a local region in the RMTP protocol. Each site has a secondary logging server. When a receiver detects a message loss, it requests a retransmission from its secondary logging server. If a secondary logging server misses a message, it contacts the primary logging server for retransmission.

The multicast sender only performs error recovery for the primary logging server. Figure 2.4 illustrates the error recovery hierarchy in LBRRM.

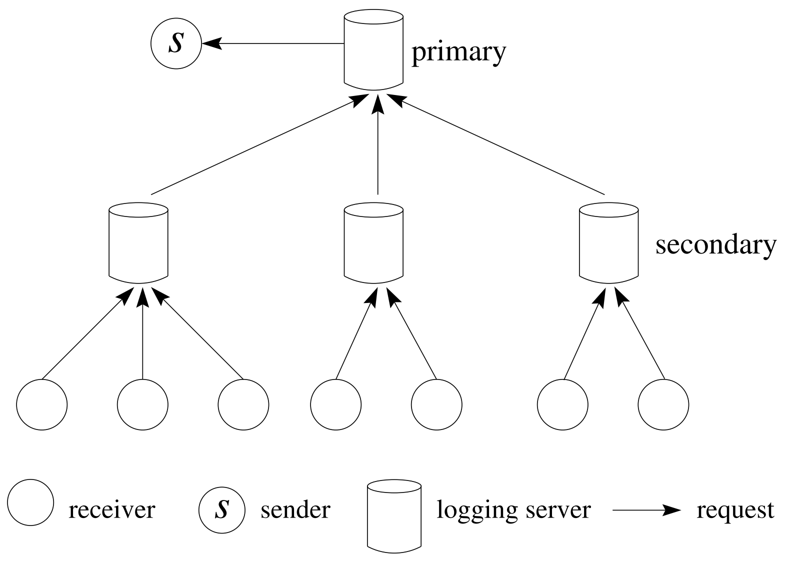


Figure 2.4: Error recovery in LBRRM is provided through a two-level hierarchy of logging servers: a primary logging server and a set of secondary logging servers.

LBRRM uses a statistical acknowledgment scheme to estimate the dissemination status of a data message and then decides whether to send the retransmission in unicast or multicast. Like RMTP, the error recovery hierarchy in LBRRM is static.

2.3.3 Tree-based Multicast Transport Protocol

The Tree-based Multicast Transport Protocol (TMTP) was designed for interactive collaborative applications [YGS95]. In this protocol, receivers are organized into a hierarchy of subnets or domains. In each domain, a domain manager is selected to perform error recovery for group members within its domain as well as for some downstream domain managers. Unlike RMTP and LBRRM, the error recovery tree in

TMTP is dynamically formed based on expanded ring search. More specifically, when a domain manager first joins the group, it multicasts a SEARCH_FOR_PARENT message with a small TTL value. If no response is received after a certain period of time, it re-multicasts the message with a larger TTL value. This process repeats until it receives a WILLING_TO_BE_PARENT message from an existing domain manager in the error recovery tree. There is, however, a trade-off between parent search time and bandwidth consumption. If the increase in TTL value is small during each try, the search process may take a long time when the group is sparse. On the other hand, if the TTL increase is large, bandwidth may be wasted for search and response messages. In addition, in this protocol, a new domain manager always chooses the closest existing domain manager as its parent, even if the parent is downstream in the underlying multicast tree. This may increase error recovery latency.

Retransmission requests and repairs in TMTP are always sent in multicast. Two optimizations are proposed to reduce bandwidth overhead due to error control messages. First, each domain manager keeps track of the TTL value to the farthest member in its domain. This is called the *multicast radius*. A domain manager keeps its members updated with the current multicast radius. When a receiver detects a message loss, it multicasts a negative acknowledgment with a TTL value equal to its multicast radius. Upon receipt of the nack, its domain manager multicasts the corresponding message with a TTL equal to the multicast radius.

It is possible that multiple receivers in a region detect a message loss at the same time. The second optimization proposed in the paper is to use randomized back-off to suppress duplicate multicasts: when a receiver detects a message loss, it waits a random amount of time before multicasting a nack. If it hears a nack from another member in its domain for the same message when it is waiting, it suppresses its own nack. This scheme is similar to the randomized back-off scheme in the SRM protocol. However, the back-off period here is proportional to the size of the local domain rather than to the size of the entire multicast group.

2.3.4 Local Group Concept

Local Group Concept (LGC) is a multicast algorithm that organizes receivers into a hierarchy of Local Groups [Hof97]. In each local group a Group Controller is selected and made responsible for processing acknowledgments for all receivers in that group. Message losses are recovered within a local group first. A Group Controller requests retransmissions from its upstream Group Controller if local recovery is not successful.

There are three types of acknowledgments in LGC: positive acknowledgment, negative acknowledgment, and semi-negative acknowledgment. A Group Controller sends a positive acknowledgment when all receivers in its local group have received the message. This indicates to the sender or a higher-level Group Controller that it is safe to discard the message. A Group Controller sends a negative acknowledgment if the entire local group missed the message. This indicates to the sender that a retransmission is needed. A Group Controller sends a semi-negative acknowledgment when some, but not all, receivers in the group have received the message. Since a message loss is recovered within a local group first, the sender does not need to retransmit the message at this moment. However, the sender cannot release the message either: if the Group Controller crashes, receivers in its local group should still be able to get the message directly from the sender.

LGC can be configured to operate in two modes: a delay-sensitive mode or a load-sensitive mode. The choice reflects a trade-off between network traffic and error recovery latency. In the delay sensitive mode, when a Group Controller receives a retransmission request, it immediately multicasts the requested message in its local group. In contrast, in the load sensitive mode, a Group Controller records the number of requests received for that message during a predefined time interval. If this number exceeds a certain threshold, it retransmits the message in multicast. Otherwise the message is sent in unicast.

The hierarchy of local groups is constructed through a Dynamic Configuration Service. In this protocol, a Group Controller periodically announces its presence by multicasting a LG_ADVERTISE message. The message contains the smoothed error probability of the Group Controller, the number of receivers in the local group, and the initial TTL value of the message[1]. A Group Controller dynamically changes the scope of its advertisement messages by choosing different TTL values. A receiver estimates its hop counts from its Group Controller to be the smallest TTL value of all advertisement messages it has received during a predefined time interval. A receiver maintains information about all Group Controllers it has heard from recently and selects the most *appropriate* one to process its acknowledgments. LGC does not propose a specific metric to measure appropriateness. Rather this decision is left to the application. One possibility is to select the closest Group Controller (in terms of hop counts) as previously done in the RMTP protocol. However, for certain applications a receiver may want to select the Group Controller which has the lowest error probability. It is also possible to consider a combination of several metrics. If a receiver cannot find an appropriate Group Controller (e.g. all existing Group

[1]It may also contain other information required by the application (e.g. carrier fees, security options).

Controllers are too far away), it may establish a new local group and elect itself as the Group Controller. Soft state timers are used to detect Group Controllers that have crashed or left the group.

2.4 Router-assisted Reliable Multicast

Recently several proposals have been made that extend the existing IP multicast service by providing new functionalities at routers. The motivation is that many of the difficulties in the design of reliable multicast protocols for large networks are due to lack of topological information about the underlying multicast tree. Such information is deliberately hidden in the IP multicast model. With assistance from network routers, reliable multicast protocols can achieve better performance and scalability. In the following, we examine two examples of router-assisted schemes: LMS [PPV98] and Search Party [CM99].

2.4.1 Light-weight Multicast Service

LMS extends the IP multicast service model with a new set of forwarding services to provide a more refined form of multicasting. In LMS, every router in the multicast tree selects one of its downstream links as the *replier link*[2]. A receiver connected to a replier link is called a *replier* and serves as a representative for performing error recovery in its subtree. Figure 2.5 shows an example of the replier assignment.

When a receiver detects a message loss, it sends a request to its router. Upon

[2]There are two exceptions. First, if a router has only one downstream link, it selects the upstream link as the replier link. Second, if a router is adjacent to the multicast sender, it selects the sender link as its replier link.

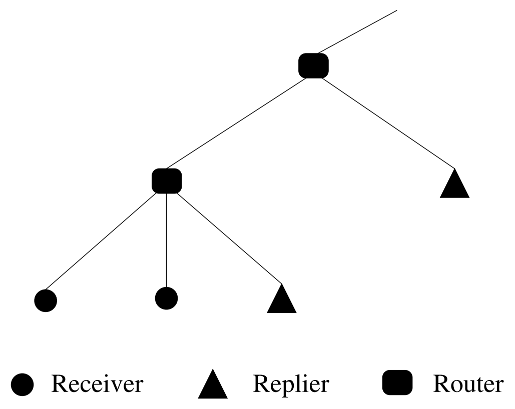


Figure 2.5: Replier assignment in LMS. Every router in the multicast tree selects one of its downstream links as the replier link. A replier serves as a representative for performing error recovery in its subtree.

receipt of the request, the router checks whether it is from its replier link. If so, the request is forwarded upstream. Otherwise the request is forwarded to the replier link. When a router forwards a request from a non-replier link to its replier link, it inserts into the request its address and the identifier of the link on which the request arrived. This router is called the *turning point* of the request. Each request can have only one turning point. The nice property of this forwarding rule is that when all receivers in a subtree miss a message, only one request is sent upstream, even though no receiver in the subtree knows exactly where the loss has occurred.

When a replier receives a request, it checks whether it has the message. If it does not have the message, it ignores the request. Otherwise it sends a reply that contains the message and the link identifier carried in the request to the router at the turning point. When the router receives the reply, it multicasts the message to the subtree rooted at the specified link. This is called a *directed multicast.* Figure 2.6 illustrates the error recovery process in LMS. On the left, every receiver in the loss subtree sends a request to its router. Only one of them is forwarded outside the loss subtree to the closest replier capable of repairing the loss. On the right, this replier sends the requested message to the turning point router, which then multicasts the message to the loss subtree.

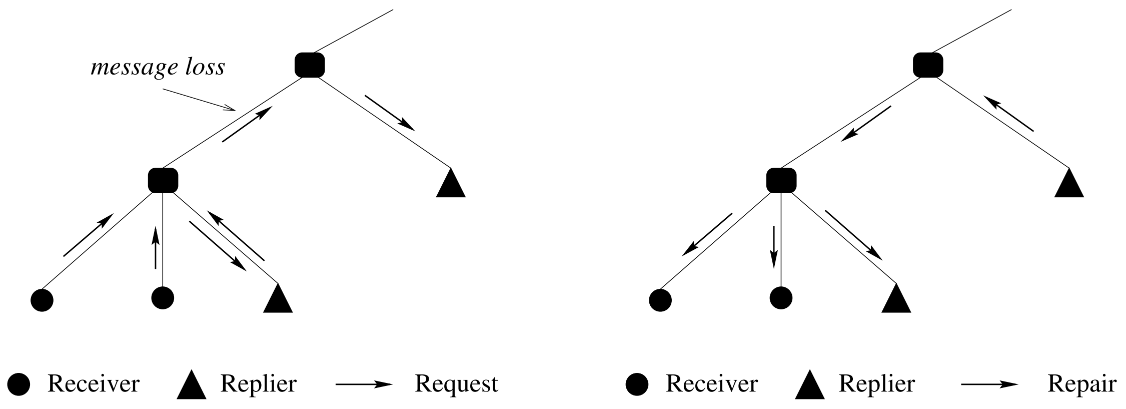


Figure 2.6: Error recovery in LMS. On the left, every receiver in the loss subtree sends a request to its router. Only one of them is forwarded outside the loss subtree to the closest replier capable of repairing the loss. On the right, this replier sends the requested message to the turning point router, which then multicasts the message to the loss subtree.

It is interesting to compare the error recovery hierarchy in LMS with that in RMTP. A replier in LMS is similar to a Designated Receiver in RMTP. Both of them serve as representatives for error recovery in a local region. However, the error recovery tree in RMTP is built at the transport layer and may not correspond well to the underlying multicast tree. With additional support from routers, LMS can have better control of error recovery traffic.

2.4.2 Search Party

Search Party is an error recovery protocol based on a new forwarding service called *randomcast*, which forwards packets randomly inside a multicast distribution tree [CM99]. In this protocol, when a receiver c detects a message loss, it sends a request in a randomcast packet to its parent node p in the multicast tree. Upon receiving the packet, p makes a random choice to decide whether to forward the packet to its parent or to another child. The probability of forwarding to p's parent is weighted by the number of leaves in the subtree under c. This, however, requires a router to keep track of the subtree population below each downstream link.

Should p decide to forward the packet to another child, it inserts sufficient information into the packet to address the subtree below the arrival interface. This allows the recipient of the request to send the repair message in a *directed multicast* that restricts its scope to the subtree rooted at the arrival link, an idea previously used in the LMS protocol. A receiver missing a message keeps sending requests as a Poisson process until a repair arrives. In effect, all receivers missing the message conduct random search inside the multicast tree. Hence comes the name *Search Party*.

Chapter 3

Bimodal Multicast

Critical applications require detailed knowledge about the behavior of their systems under various operational conditions. Some of these applications also produce a stream of data that needs to be delivered at a steady rate. For example, in an air traffic control setting, periodic updates to radar images and flight tracks need to be communicated to a group of controllers in a timely manner. Stability in message delivery is essential to safe operation.

Traditionally, multicast protocols designed for such applications provide strong reliability properties such as atomicity (if a multicast is delivered to any correct process, eventually it will be delivered to all correct processes), total ordering delivery, and virtual synchrony. Providing these properties entails costly protocols and may trigger erratic throughput in demanding environments.

This chapter describes a new approach to reliability for critical applications: a Bimodal Multicast protocol [BHO$^+$99] that has superior scalability and strong throughput properties. Unlike virtual synchrony protocols that provide an “all or nothing” guarantee, Bimodal Multicast protocol provides a bimodal delivery guarantee: for any multicast message sent to the group, there is a high probability that the message will reach almost all members, a small probability that the message will reach a small number of members, and a vanishingly low probability that the message will reach *many* but not *all* members. Experimental results demonstrate that the protocol achieves steady throughput even when failures occur.

The rest of the chapter is organized as follows. Section 3.1 describes the error recovery algorithm in Bimodal Multicast. Section 3.2 describes two optimizations of the basic algorithm that improve its performance under network failures. Section 3.3 analyzes the performance of the protocol using formal methods. Experimental results are presented in Section 3.4. Section 3.5 concludes.

3.1 Gossip-based Anti-entropy Protocol

Bimodal Multicast employs a gossip-based anti-entropy protocol for error recovery. The idea of gossip was previously used in epidemic algorithms to disseminate updates in a distributed database environment [OD81, DGH$^+$87]. More recently, van Renesse et al. proposed a failure detection service using the random gossiping technique [vRMH98]. Other work on gossip protocols include [LOM94, For95, LLSG92, GT92].

The protocol proceeds through a series of rounds. The length of each round must be larger than the round trip time between any two members in the group. In the current implementation, a round starts every 100ms. During each round a member p sends its history of received messages to a randomly selected member q in the group.

This message is called a *gossip* message. Upon receiving the message history, q asks p for any message that q missed but p has received so that p and q can converge to identical histories. Figure 3.1 illustrates this process in a group of four members, one of which is the sender. The sender multicasts a series of messages unreliably to the group. Member r missed message 2. During the subsequent gossip round, p sends a summary of its received messages to r (the gossip message). Upon receipt of the summary, r discovers that it has missed message 2 and then solicits a retransmission from p.

If a member discovers that it has missed several messages, it solicits the most recent message first. For example, if member r missed messages 1, 2, 3, and 4, it would request a retransmission of message 4 first, followed by messages 3, 2, and 1. Hence the protocol emphasizes achieving a common suffix of message history rather than a common prefix. For applications where data are periodically refreshed (e.g. stock quotes in a stock market environment), new messages are usually more important than old messages.

After a member receives a message, it buffers the message for a fixed number of rounds and then garbage collects the message. In the current implementation, a member garbage collects a message 10 rounds after its initial reception. Because of the "most-recent-first" retransmission strategy and the randomized nature of the algorithm, it is possible that a message is still missing at some member but has been garbage collected everywhere else. If a member cannot recover a message loss after a certain amount of time, it gives up on the message and reports the loss to the application. Hence the protocol is suitable for applications that have a strong emphasis on stable throughput but can tolerate infrequent message loss.

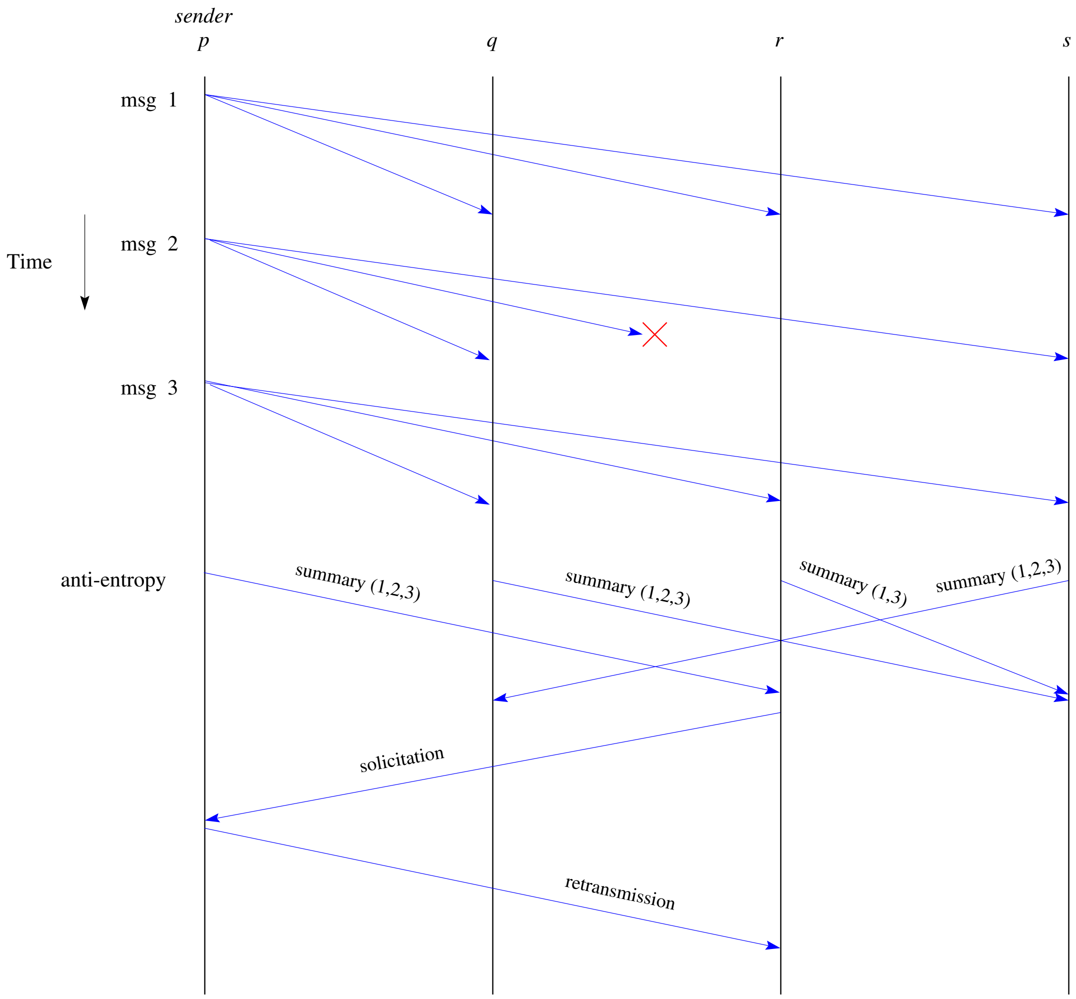


Figure 3.1: Error recovery in Bimodal Multicast. Members in the group periodically exchange history of received messages. When r receives message history from p, it discovers that it missed message 2 and hence solicits a retransmission from p.

3.2 Optimization

There are several optimizations of the basic gossip protocol that aim to improve its performance under network failures. This section describes two such optimizations. Their effectiveness is evaluated in Section 3.4. A complete description of all optimizations can be found in [BHO+99].

Optimization I: Round Retransmission Limit. This optimization restricts the maximum amount of data a process will retransmit in any given round. As soon as a process has reached the limit, it ignores subsequent solicitations until the next round. This avoids a situation where a poorly performing process puts a heavy load on a healthy process by swamping it with retransmission requests. If a process has fallen far behind the group, the load of error recovery for this process will spread across several rounds among different processes.

Optimization II: Multicast for Some Retransmissions. In the basic gossip protocol, a process always sends a retransmission in unicast. However, this may incur a high error recovery latency for system-wide message loss. In the extreme case where the sender experienced a send omission failure, only the sender holds a copy of the message initially. All other receivers need to recover the message loss through the gossip protocol. Epidemic theory shows that the expected time for a receiver to recover a message loss in this case is proportional to the log of the group size [Pit87, DGH+87]. Better efficiency can be achieved if a process uses a mixture of unicast and multicast for retransmissions. In this optimization, a receiver keeps track of the number of retransmission requests it has received for a message. If this number exceeds a prespecified threshold, it multicasts the retransmission to the entire group. The fact that many requests for a message have been received is a good indication that the initial IP multicast failed.

3.3 Analysis

In this section, we analyze the performance of the protocol using formal methods. We compute the distribution of latency between when a message is sent and when it is delivered. For our analysis, we consider just a single multicast message in a group with a fixed set of processes. These processes communicate with each other over a fully connected, point-to-point network. Initially one process multicasts the message unreliably to the whole group. This message is received by a random subset of processes. These processes include information about this message in subsequent gossip messages they send. Each process independently chooses the destination for its gossip message uniformly at random from all other processes in the group. For simplicity, we combine solicitations and retransmissions into a single gossip message.

The protocol executes in a series of rounds in which messages sent in the current round are delivered in the next. We consider two types of failures in the analysis: process failures and message failures, both benign in nature. There is an independent probability of at most τ that a process crashes during one round. There is also an independent, per-process probability of at most ϵ that a gossip message sent between nonfaulty processes is lost in the network. Since the failure of a gossip message implies either the failure of the solicitation or the failure of the retransmission, $1 - \epsilon$ is the probability that the message gets through back and forth. In practice we expect both ϵ and τ to be small. We assume that message failures and process failures are independent of each other. We do not consider Byzantine failures.

In the following analysis we derive a recurrence relation showing how the number of processes that have yet to receive a copy of the message decreases over time. We define the following parameters:

N : the number of processes in the system.

ϵ_0 : the independent probability that a correct process (other than the sender) does not get a message during its initial multicast.

β : the probability that a process gossips to every other process.

s_t : the number of processes that may gossip in round t (or in epidemic terminology the *infectious* processes).

r_t : the number of processes in round t that have not received a gossip message yet (the *susceptible* processes).

Since the sender always has a copy of the message, initially we have:

$$\begin{aligned} s_0 &= 1 + (N-1) \cdot (1-\epsilon_0) \\ r_0 &= (N-1) \cdot \epsilon_0 \end{aligned}$$

$\epsilon_0 = 1.0$ corresponds to the case where the initial multicast fails and only the sender has a copy of the message at the outset. In this case, we have: $s_0 = 1$ and $r_0 = N-1$. Given s_t and r_t, we derive a recurrence relation for s_{t+1} and r_{t+1}. First we assume that processes do not crash. We introduce constants

$$\begin{aligned} p &= \beta \cdot (1-\epsilon) \\ q &= 1-p \end{aligned}$$

p is the probability that both an infectious process gossips to a particular susceptible process and that the message gets through. For each of the r_t susceptible processes, we consider the probability that at least one of the s_t infectious processes sends a gossip message that gets through. Expressed differently, this is the probability that *not* all infectious processes fail to send a message to a particular susceptible process:

$$1 - (1-p)^{s_t} = 1 - q^{s_t}$$

Let k be the expected number of newly-infected processes in round t. We then have

$$\begin{aligned} k &= r_t \cdot (1 - q^{s_t}) \\ s_{t+1} &= s_t + k \\ r_{t+1} &= r_t - k \end{aligned}$$

Now we introduce process failures into the analysis. There is an independent, per process probability of τ that a process has a crash failure during one round of the protocol. There are three ways in which processes can fail. They can crash before, during, or after the gossip stage of the Bimodal Multicast protocol. Here we are investigating the relationship between the number of susceptible processes and the number of gossip rounds. The worst case occurs when all faulty processes crash before the gossip stage. In this case the probability for a susceptible process to receive a gossip message is equivalent to the case where there were $s_t \cdot (1-\tau)$ correct processes gossiping at the beginning of the round. Similarly, we can relax the message failure rate to be anywhere in the range of $[0..\epsilon]$, but the worst case occurs when the failure rate is ϵ. We now have:

$$s_t = s_t * (1 - \tau)$$

$$\begin{aligned} r_t &= r_t * (1 - \tau) \\ k &= r_t * (1 - q^{s_t}) \\ s_{t+1} &= s_t + k \\ r_{t+1} &= r_t - k \end{aligned}$$

Let $N = 1000$, $\beta = 1/(N-1)$ (only gossip to one other process), $\tau = 0.001$, and $\epsilon = 0.05$. Figure 3.2 shows how the number of susceptible processes decreases as a function of the number of gossip rounds. The top figure shows the case where the initial IP multicast fails (i.e. $\epsilon_0 = 1.0$). The bottom figure shows the case where the initial IP multicast reaches 90% of the processes (i.e. $\epsilon_0 = 0.1$). The scales of the two figures are different.

Now we continue the above analysis to compute the expected number of rounds before a selected correct participant receives a multicast. Define:

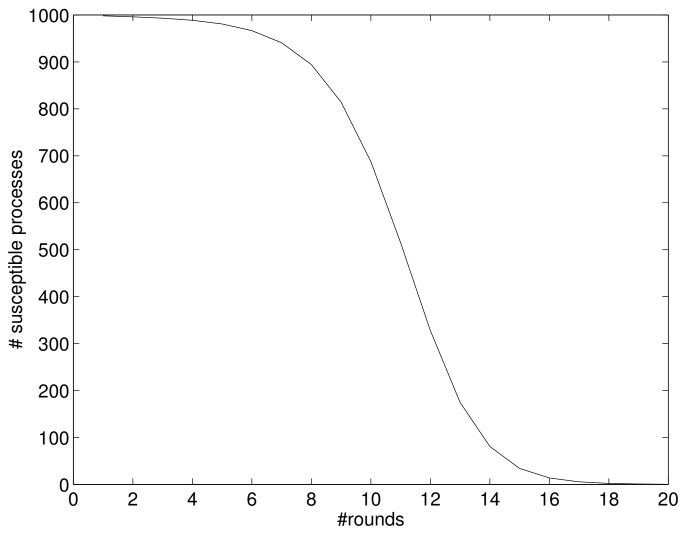
v_t : the probability that a susceptible process gets infected up to round t.

w_t : the probability that a susceptible process gets infected in round t.

Observe that in any given round all the currently susceptible processes have equal probability of getting infected. We have:

$$\begin{aligned} v_t &= 1 - \frac{r_t}{N} \\ w_1 &= v_1 \\ w_t &= v_t - v_{t-1} \end{aligned}$$

From this we are able to produce Figure 3.3 (assume that the initial IP multicast fails), showing the probability for a correct process to receive a message in a certain



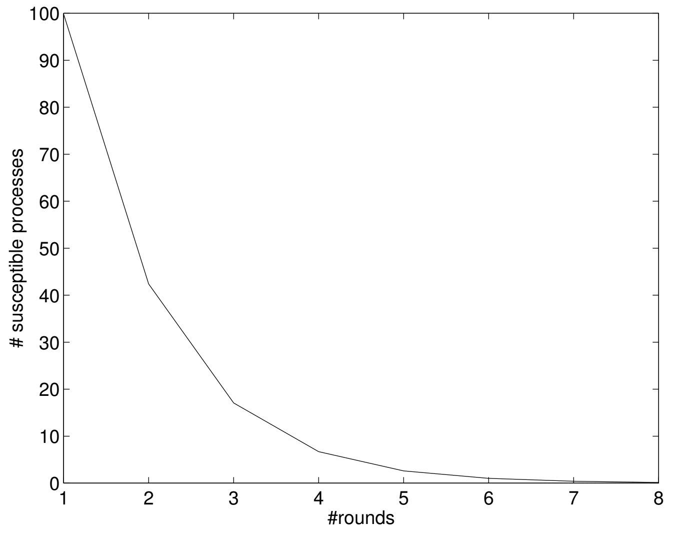


Figure 3.2: Number of susceptible processes versus number of gossip rounds when the initial multicast fails (top) and when it reaches 90% of processes (bottom). Both runs assume 1000 processes. The scales of the two figures are different.

round. The figure superimposes curves for various values of N. They have roughly the same shape with peak at roughly the log of the group size.

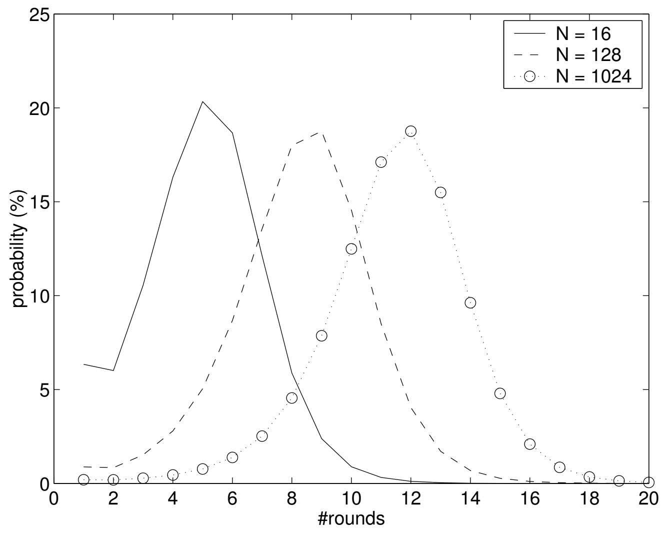


Figure 3.3: The probability for a correct process to receive a message in a particular round for groups of various sizes. Assume that the initial IP multicast fails.

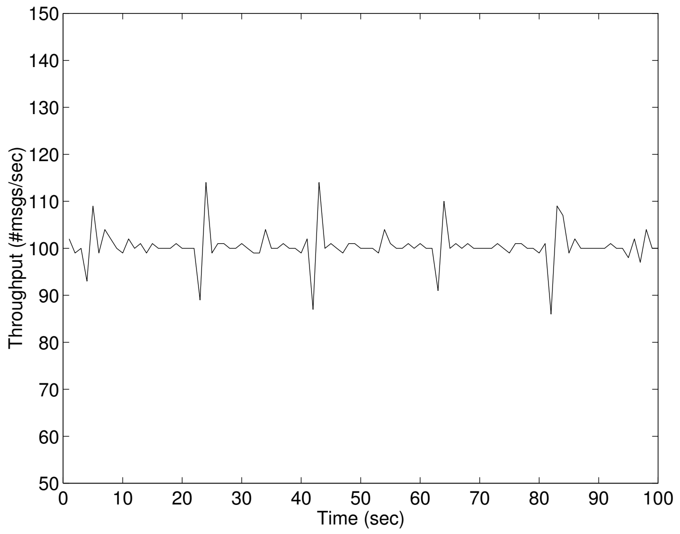
3.4 Experimental Results

This section presents experimental results of the Bimodal Multicast protocol on a local-area network. The tests are conducted in a group of 35 processes connected by 10Mbit Ethernet. One of them is the sender. The sender is sending $1K$ byte messages at a rate of 100 messages per second. Messages are delivered to the application in FIFO order. We design two sets of experiments to explore the impact of the two optimizations described in Section 3.2. The first optimization puts a limit on the amount of data a process will retransmit in any given round. In the following experiment, this limit is set to $10K$ bytes. Since each message is $1K$ bytes, this restricts the maximum number of retransmissions to 10 messages. To test the effectiveness of this optimization, we introduce a burst of message loss every 20 seconds by letting 30% of the processes simultaneously discard 50 consecutive messages. These processes will solicit retransmissions during subsequent gossip rounds. We measure the impact of this perturbation on a healthy receiver. The results are shown in Figure 3.4.

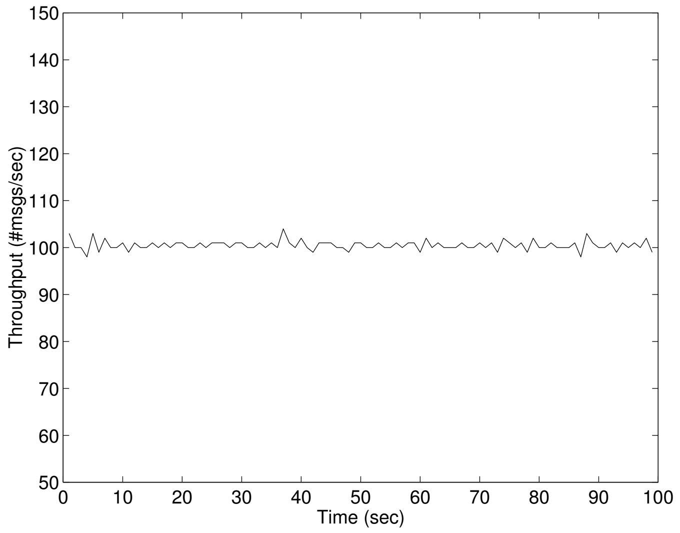
As can be seen from the figure, without the round retransmission limit (the top figure), the perturbation we introduced caused periodic fluctuation in throughput. This is because receivers lagging behind tried to catch up on all the lost messages at once and hence put a heavy load on healthy receivers. In contrast, with the optimization (the bottom figure), throughput at a healthy receiver is fairly steady. There is, however, a trade-off between the performance of healthy receivers and that of perturbed receivers. On one hand, imposing a round retransmission limit can protect healthy receivers from being dragged behind by perturbed receivers. On the other hand, a perturbed process will have more trouble catching up. For certain applications, maintaining steady throughput at healthy receivers is more important.

The second optimization described in Section 3.2 involves using multicast recovery for correlated communication failures. In the following experiment, the multicast threshold is set to 2. That is, the second time a receiver gets a request for a message, it multicasts the message in the group. To test the effectiveness of this optimization, we introduce a burst of system-wide message loss every 20 seconds by emulating send omission failures at the sender for 10 consecutive messages. We measure the throughput at one receiver to see the impact of different retransmission strategies. The results are shown in Figure 3.5.

When retransmissions are always sent in unicast (the top figure), the throughput

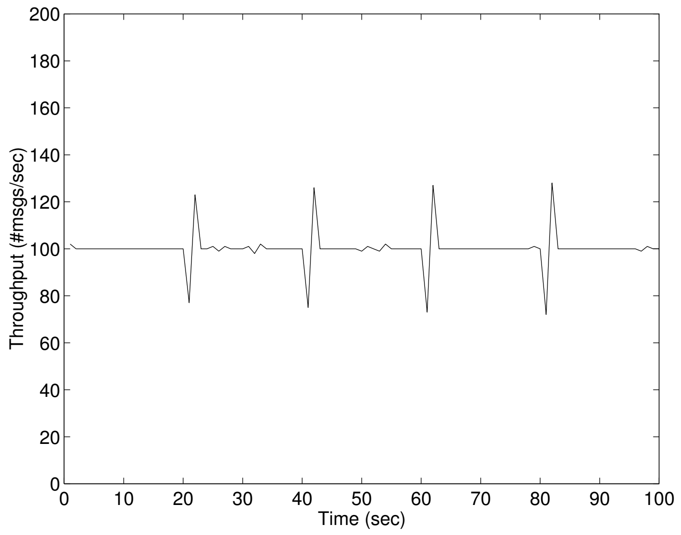


(a) Without Round Retransmission Limit

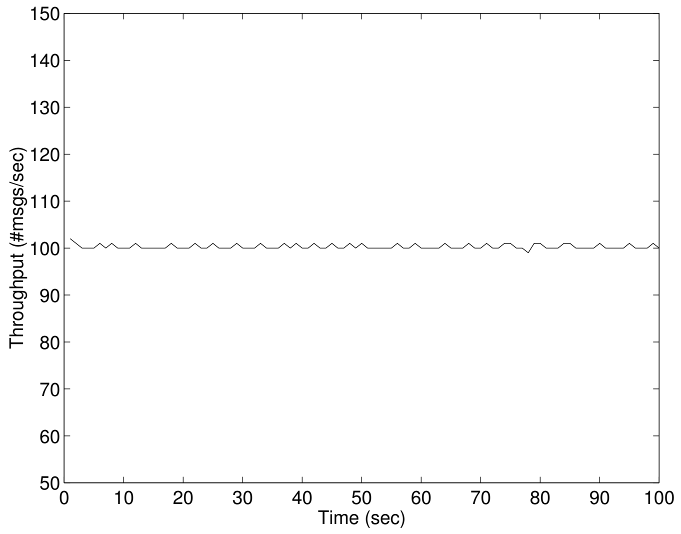


(b) With Round Retransmission Limit

Figure 3.4: Effectiveness of round retransmission limit on throughput at healthy members.



(a) Without multicast retransmissions



(b) With multicast retransmissions

Figure 3.5: Effectiveness of multicast recovery for correlated communication failures.

demonstrates periodic fluctuation each time a send omission failure occurs. This is due to the long error recovery time: in order to achieve FIFO order delivery, subsequent messages are held up in a receiver's buffer and then delivered to the application in bursts. In contrast, the throughput is very stable when a mixture of unicast and multicast is used for retransmissions (the bottom figure). These results demonstrate that selective use of multicast for retransmissions can significantly reduce error recovery time when the initial IP multicast fails.

3.5 Conclusion

Stable throughput is a desirable property for applications that require high performance and scalability. This chapter has described a Bimodal Multicast protocol developed in part by the author that uses random gossiping technique to achieve strong throughput guarantees. Both analysis and experimental results demonstrate that the protocol has good performance in demanding environments.

Although Bimodal Multicast emerged from a group effort, this chapter focuses on aspects for which the author was the primary contributor. This includes the multicast retransmission strategy described in Section 3.2, the analysis of delivery latency in Section 3.3, and the experimental results presented in Section 3.4. A complete description of the protocol can be found in [BHO$^+$99].

Chapter 4

RRMP: A Randomized Reliable Multicast Protocol

For multicast applications with only one sender, several tree-based protocols have been proposed as an efficient way to avoid message implosion and to provide good local recovery[1]. In these protocols, receivers are grouped into local regions based on their geographic proximity. A repair server is selected in each region and made responsible for performing error recovery for all receivers in that region. Because a message loss can be repaired by a regional repair server rather than by the sender, this scheme reduces recovery latency and avoids message implosion at the sender. Some examples of tree-based protocols are described in Chapter 2.

However, we also discovered several performance problems with a tree-based protocol. The first problem is that a tree-based protocol does not eliminate message implosion. Although it avoids global message implosion at the sender, each region

[1]Multiple senders can be supported by building a separate error recovery tree for each sender.

still suffers from a *regional implosion* problem because the responsibility of error recovery in each region is concentrated on a single repair server. For large regions, a repair server may be overwhelmed with retransmission requests when all receivers in its region miss a message. One way to reduce the load on a repair server is to put a limit on the size of a local region. If a region becomes too large, it is split into several small ones. This approach is effective if all receivers in the region have roughly the same loss rate. Otherwise a single receiver suffering a high loss rate can put a heavy burden on its repair server even after the split and may affect all other receivers in its region. In Chapter 7 we quantify the problem of load concentration on repair servers using simulation.

The second problem with a tree-based protocol is that the failure of a repair server can lead to a disruption of service in its region. Because a repair server is the only process capable of performing error recovery for a local region, if the server fails, error recovery in its region cannot proceed until a new server is selected. Some tree-based protocols use soft-state timers to enhance robustness. In RMTP, for example, if a receiver has not heard from its repair server for a certain period of time, it concludes that the server has failed and switches to the server least upstream from the failed server as its new repair server. This is illustrated in Figure 2.3 of Chapter 2. We feel that this is not a perfect solution for two reasons. First, failure detection in a distributed environment is a difficult problem in its own right. In practice, many systems use a conservative timer to avoid false alarms. Before the failure of a repair server is detected, receivers in its region may continue sending retransmission requests to this failed server. Second, switching to a repair server in an upstream region defeats the purpose of building an error recovery hierarchy because message loss is no longer recovered locally. Relying on a repair server in a remote region not only introduces extra traffic on wide area links but also increases error recovery latency.

The third problem with a tree-based protocol is that its performance depends on the positions of repair servers. To maximize the possibility of local recovery, a repair server should experience the least amount of message loss among all receivers in its region. Hence the optimal position of a repair server is at the root of the multicast subtree of its region. However, the topological structure of the underlying multicast tree can change dynamically due to various network conditions and is generally not known at the transport layer. Improper positioning of repair servers can lead to poor locality in error recovery. Figure 4.1 shows a situation where the repair server q in one local region is placed at the left branch of the multicast subtree in its region rather than at the root. A message loss along the indicated link causes all receivers at the left branch (including the repair server) to miss the message. Ideally, this loss can be repaired by the two local receivers at the right branch of the tree. However, in a tree-based protocol like RMTP, the repair server will request a retransmission from its upstream repair server p outside this local region, leading to long recovery latency[2].

In this chapter, we propose a randomized reliable multicast protocol called RRMP which eliminates the message implosion problem and provides good local recovery. This protocol is built on our previous work with the Bimodal Multicast protocol described in Chapter 3, but focuses on how to use randomization to improve the robustness and efficiency of tree-based protocols. A detailed comparison between

[2]A similar problem can happen even if the repair server is at the root of the tree, because the server can miss a message due to a local buffer overflow.

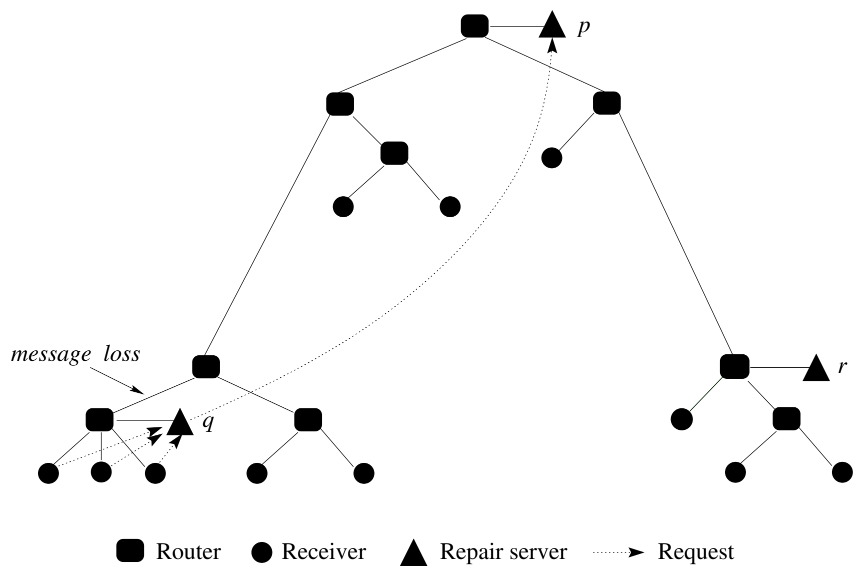


Figure 4.1: Performance penalty in a tree-based protocol when the position of a repair server is suboptimal.

RRMP and Bimodal Multicast is deferred to Chapter 8 so that sufficient background can be established. This protocol has been described in part in [XB01a, XB01b].

RRMP groups receivers into a hierarchy, similar to the tree-based protocols. Unlike those protocols, RRMP lets each receiver that experiences a message loss send its request to randomly selected receivers in its local region and, with a small probability, to some randomly selected receiver in a remote region. The reliability of the protocol depends on statistical properties of its randomized algorithm which can be formally analyzed and tuned according to application requirements. Simulation results in Chapter 7 demonstrate that the protocol achieves fast error recovery with low overhead, compared to tree-based protocols.

The rest of the chapter is organized as follows. Section 4.1 describes the system model and assumptions made in the protocol. Section 4.2 describes the details of the error recovery algorithm. Section 4.3 presents a scheme for measuring round trip time between receivers in the group. Section 4.4 describes two optimizations of the basic algorithm to reduce error control traffic. Section 4.5 analyzes its performance. Section 4.6 concludes.

4.1 System Model and Assumptions

We consider multicast applications with only one sender. The sender joins the multicast group before it starts sending messages, and consequently is also a receiver in the group. We assume that receivers are grouped into local regions and that different regions are organized into a hierarchy according to their distance from the sender. We call this the *error recovery hierarchy.* Figure 4.2 shows an example of a hierarchy where the whole group is divided into three local regions. Chapter 5 describes an algorithm for constructing such a hierarchy. We define the *parent* region of a receiver as its least upstream region in the hierarchy[3]. For example, in Figure 4.2, region 1 is the parent region for all receivers in region 2. If a receiver is in the same region as the sender, then it has no parent region. Hence none of the receivers in region 1 has a parent region. We also assume that each receiver maintains group membership knowledge about other receivers in its region as well as receivers in its parent region. Chapter 5 explains how a receiver can obtain such information.

A receiver detects a message loss by observing a gap in the sequence number space or by exchanging session messages with other members (described in Chapter 5). In this chapter, we focus on the error recovery algorithm in RRMP. Other aspects of

[3]This definition will be refined in Chapter 5 where we show that a receiver's parent region may not necessarily correspond to a local region.

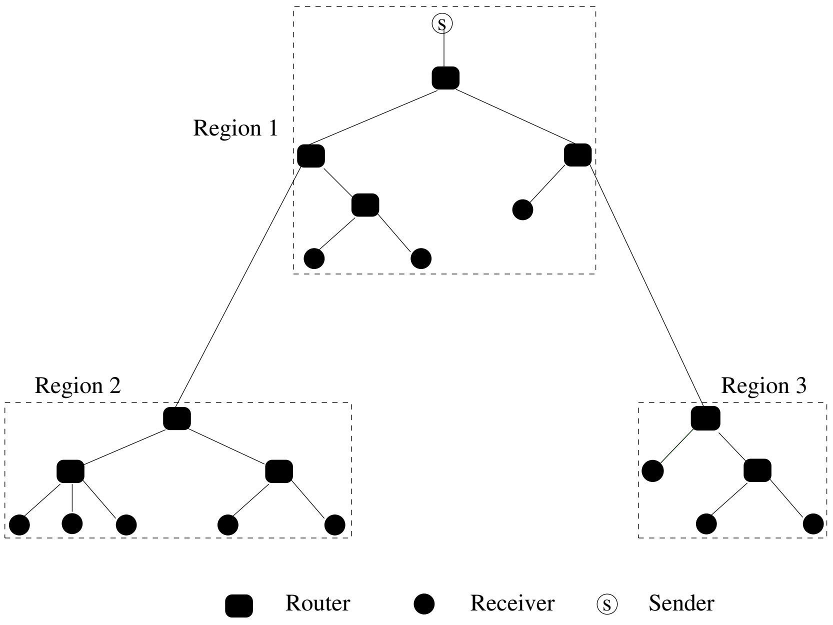


Figure 4.2: Local regions in a hierarchical structure

the protocol (e.g. formation of the error recovery hierarchy, buffer management) are described in subsequent chapters.

4.2 Details of the Algorithm

Unlike tree-based protocols, RRMP does not use any repair server. Responsibility for error recovery is randomly distributed among all receivers in the group. Assume that a receiver p detected a message loss. The loss can either affect a fraction of receivers in p's region (a *local* loss), or can affect all receivers in that region (a *regional* loss). In the first case, p can receive a retransmission from neighbors in its region, while in the second case the loss can only be repaired by a member in a remote region. Accordingly, the error recovery algorithm in RRMP consists of two phases executed concurrently: a local recovery phase and a remote recovery phase. A local loss can be repaired through local recovery, while a regional loss is repaired by a combination of local recovery and remote recovery. The rest of the section describes the details of the two recovery phases.

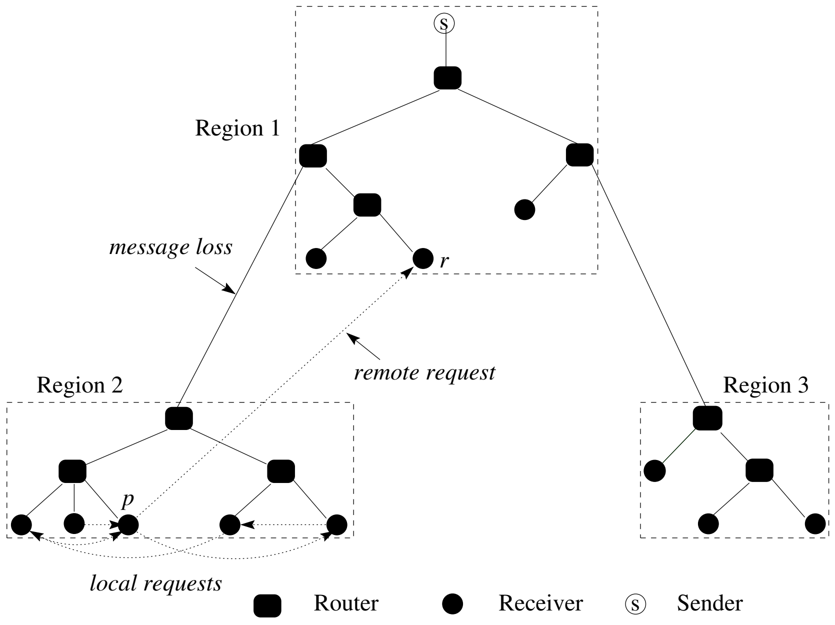
In the local recovery phase, a receiver tries to recover a message loss from randomly selected neighbors. More specifically, when a receiver p detects a loss, it selects a receiver q uniformly at random from all receivers in its region and sends a request to q. p also sets a timer according to its estimated round trip time to q. Round trip time measurements are described in the next section. Upon receiving p's request, q checks whether it has the message. If so, it sends the message to p. Otherwise it ignores the request. If p does not receive a copy of the message when its timer expires, it randomly selects another receiver in its region and repeats the above process. As long as at least one local receiver has the message, p is eventually able to recover the lost message. This has been shown in previous work on epidemic theory [Bai75, OD81]. In particular, a receiver in the sender's region is able to recover any lost message through local recovery.

On the other hand, if an entire region missed a message, the message loss cannot be repaired within the local region. In tree-based protocols, the repair server of the region is responsible for contacting a remote member for retransmission. In RRMP, this responsibility is taken by some randomly selected members in the region during the remote recovery phase. More specifically, when a receiver p detects a message loss, it randomly chooses a remote receiver r in its parent region and, with a small probability P, sends a request to r. P is chosen so that the expected number of remote requests sent by all receivers in the region is a constant λ. For example, let

n be the number of receivers in a region. If $P = 1/n$, then on average one remote request is sent when the entire region missed a message (hence $\lambda = 1$). p also sets a timer according to its estimated round trip time to r. This timer is set by any receiver missing a message, regardless whether it actually sent out a request or not. If p does not receive a copy of the message when its timer expires, it randomly selects another receiver in its parent region and repeats the above process. As long as the entire region misses the message, the expected number of remote requests during each try is λ.

Upon receiving a request from a remote receiver p, r checks whether it has the requested message. If so, it sends the message to p. Otherwise, r missed the message as well. In this case, r records "member p is waiting for the message". When r later receives a copy of the message, it will relay the message to p. Since r's region is upstream of p's region, it is likely that r will detect and recover the lost message earlier than p. When p receives a repair message from a remote member, it checks whether the message is a duplicate. If not, p multicasts the message in its local region so that other members sharing the loss can receive the message.

The two phases described above, local recovery and remote recovery, are executed concurrently when a receiver detects a message loss (the receiver does not know how many members in its region missed the same message). If a receiver has no parent region, its remote recovery phase does nothing. Figure 4.3 illustrates RRMP's error recovery algorithm when all receivers in region 2 missed a message. On the top, local requests are sent to randomly selected neighbors, and one of them, p, sends a request to a remote member r. On the bottom, member r forwards a copy of the message to p, which then multicasts the message in its local region.



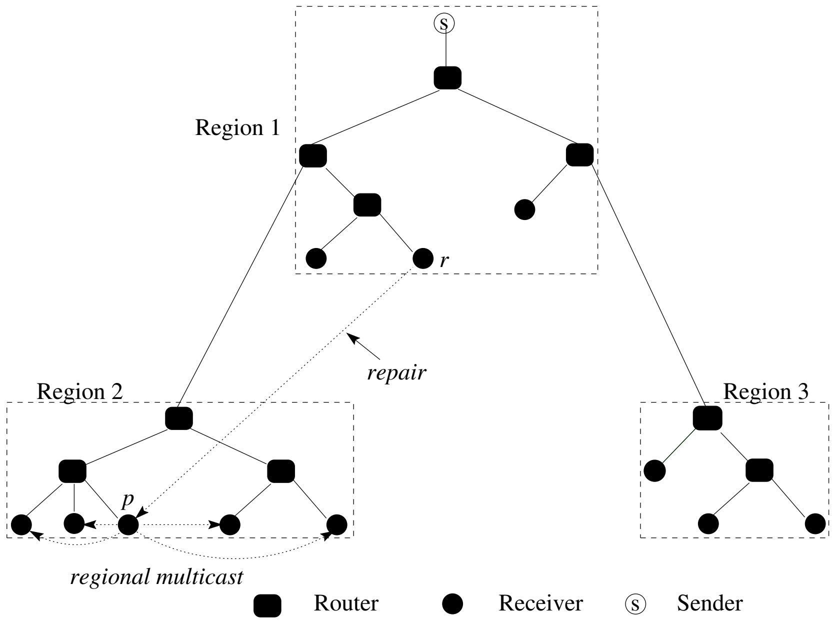


Figure 4.3: Error recovery in RRMP

Figure 4.4 shows another example where the error recovery hierarchy has three levels. For simplicity, only remote requests are shown in the figure. On the top, a message loss as indicated caused all receivers in region 2 and region 4 to miss the message. Receivers in region 2 first detect the message loss. One of them, q, sends a remote request to an upstream member s. Later, receivers in region 4 detect the message loss. One of them, p, sends a remote request to r. When r receives the request from p, it finds itself missing the same message. In this case, it records p's request. On the bottom, s sends a copy of the message to q. When q receives the message, it multicasts the message in its local region. Now r receives the message. Because it remembers p's request, it forwards the message to p. Finally, p multicasts the message in its local region.

4.3 Round Trip Time Measurements

Our error recovery algorithm requires that a receiver measure its round trip time (RTT) to its neighbors as well as to receivers in its parent region. This is achieved by attaching timestamps in request and repair messages[4]. Measuring RTT to a local member is straightforward: when a receiver p sends a request to a local member q, it attaches a timestamp to the request. This timestamp is copied onto the repair message sent by q (assume that q has the requested message). Upon receiving the repair, p can compute its RTT to q using a TCP-like scheme [Jac88].

Measuring RTT to a remote member is more complicated because a receiver does not always send a repair immediately after receiving a remote request. This is the

[4]Clocks at different receivers need not be synchronized.

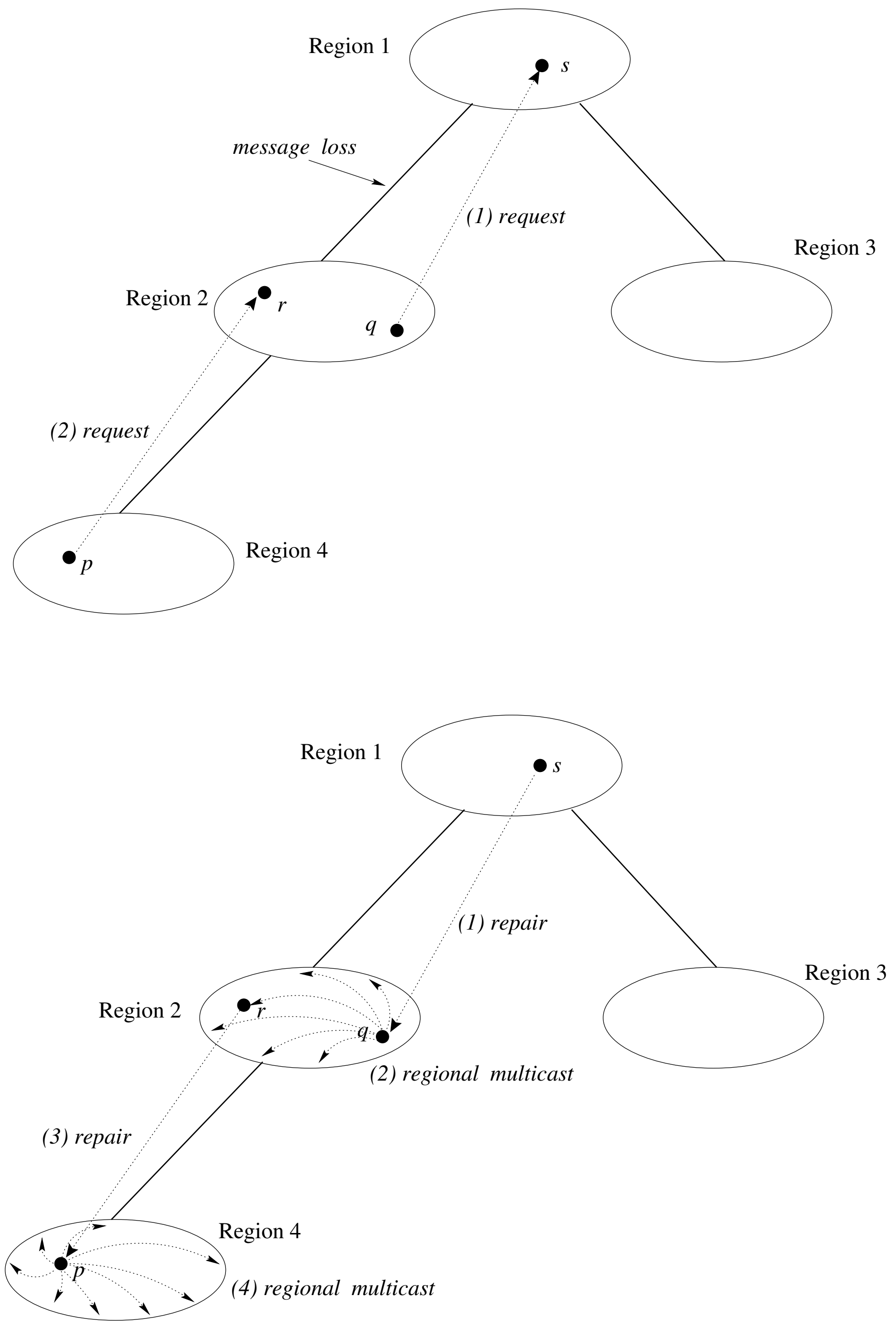


Figure 4.4: Error recovery in a multi-level hierarchy

case if the receiver is waiting for a repair for the same message. RRMP addresses this problem by letting the receiver include local processing time in the repair message, an idea previously used in the Network Time Protocol [Mil91]. More specifically, when a member r sends a repair to a remote member p, it includes two timestamps, t and Δ, where t is the timestamp copied from p's request, and Δ is the interval between the time r received p's request and the time r sent the repair. Upon receiving the repair, p can compute its RTT to r by excluding r's local processing time. This process is illustrated in Figure 4.5. Moreover, p also includes its RTT estimation to r when it multicasts the repair message in its region. In a wide area network, the latency between two regions can be much higher than the latency within a region. Hence all members in a region can share the same RTT estimation to a remote member.

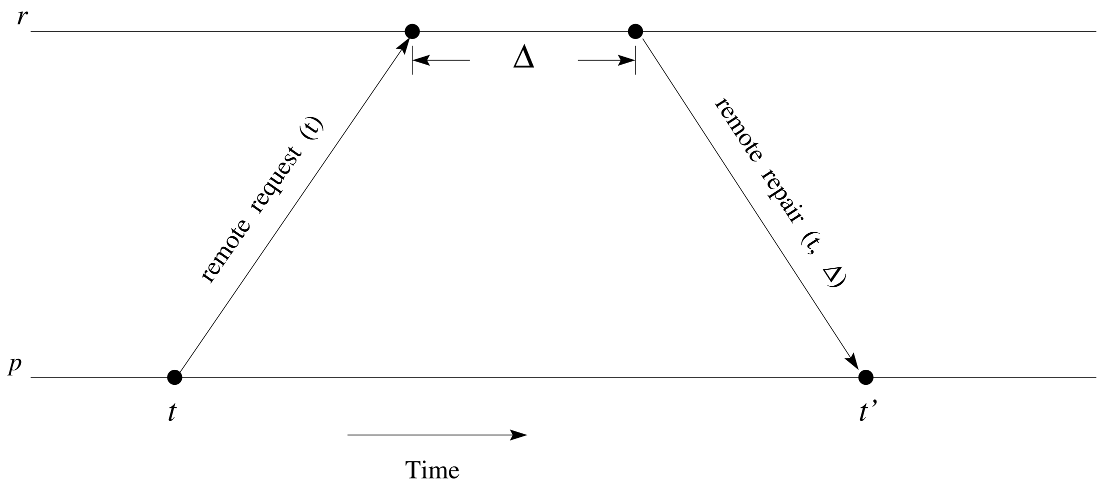


Figure 4.5: Round trip time measurement to a remote member. p's RTT to r is: $t' - t - \Delta$.

The accuracy of RTT estimation depends on the frequency of requests and repairs sent. To maintain reasonable accuracy during periods when system loss rate is low, each receiver enforces a maximum interval T_{maxl} between any two local requests and a maximum interval T_{maxr} between any two remote requests. If no message is lost, the receiver sends a special request, *RTT_QUERY*, at the end of the interval, which triggers an immediate *RTT_RESPONSE* message. The choice of T_{maxl} and T_{maxr} reflects a trade-off between bandwidth consumption and the accuracy of RTT measurements.

Round trip time estimation in the presence of network dynamics is a complicated problem and has received much attention in the literature [Pax97b, Pax97a, AP99]. Recent studies indicate that Internet RTTs resemble a heavy-tailed distribution with occasional spikes of extraordinarily high values [AP99]. Most of these studies focus on the accuracy of RTT estimation in the TCP protocol. It has been shown that many TCP implementations use very conservative timeout values due to the high penalty of false alarms: a premature timeout not only triggers an unnecessary retransmission but also causes the congestion control algorithm in TCP to enter the slow start phase [Pax97b]. In contrast, a bad timeout has less profound impact on RRMP.

4.4 Optimization

In this section, we describe two optimizations of the basic error recovery algorithm. The first optimization aims to reduce request traffic, while the second optimization aims to reduce repair traffic.

4.4.1 Reducing Request Traffic

During the local recovery phase, a member missing a message sends requests to randomly selected neighbors. However, if the message loss is regional, it can only be repaired by a remote member. If inter-region latency is much higher than intra-region latency, it is inefficient for a member to keep sending local requests until the loss is repaired. Consider the topology in Figure 4.3. Assume that p's RTT to r is 100ms and to a local member is 1ms. In this case p could send approximately 100 local requests before it gets a repair from r.

The amount of request traffic can be reduced if a member stops sending local requests when it can conclude with high confidence that no member in its region has the message. For example, for a region of 30 members with one member holding the message initially, the probability that a member missing the message will receive a repair within 7 requests is 99.5% (this can be calculated by summing up the probability in Figure 4.7 described in the next section). Hence if a loss is not repaired after sufficiently many local requests were sent, the member can stop its local recovery phase because it is highly likely that the entire region has missed the message. When some member in the region later receives a repair during its remote recovery phase, it will multicast the repair in the region. This multicast, however, is also subject to loss and may fail to reach some member. Hence, if a member stops its local recovery phase forever, it will not be able to repair the loss locally when it can (because now some members do have the message). To avoid this situation, a member restarts its local recovery phase whenever the timer in its *remote* recovery phase expires. With this optimization, error recovery in RRMP consists of a single remote recovery phase and a series of local recovery phases. Each local recovery phase is triggered by a timeout during the remote recovery phase except the first one, which starts upon detection of the loss.

Figure 4.6 illustrates this optimization when all three receivers in a downstream region missed a message. We first look at the top figure. Each of the three receivers starts the local recovery phase and the remote recovery phase simultaneously. During the remote recovery phase, p sends a request to an upstream member r. During the local recovery phase, all three receivers send requests to randomly selected neighbors. However, because they all missed the same message, these requests never trigger any repair. After a while, they conclude that the message loss is regional and hence stop sending local requests. Later, when p gets a remote repair from r, it multicasts the message in its local region so that the other two receivers can get the message.

The bottom figure shows a situation similar to the previous one except that the regional multicast sent by p fails to reach receiver q. In this case, q restarts its local recovery phase when the timer in its remote recovery phases expires and gets a retransmission from p.

4.4.2 Reducing Repair Traffic

During the remote recovery phase, when all members in a region missed a message, on average λ of them will send remote requests. When a member receives a repair from a remote member, it multicasts the repair in its region if the repair is not a duplicate. Hence if two members receive a repair at the same time, both of them will multicast the repair. The number of duplications in this case is independent of n but increases with λ. In order to reduce the number of duplicate repairs, we employ a randomized back-off scheme to suppress duplicate regional multicasts at the expense of potentially longer recovery latency: upon receiving a remote repair, a member makes a random decision as whether it should multicast the repair. The probability it does so is $1/\lambda$. Otherwise it waits a random amount of time and tries

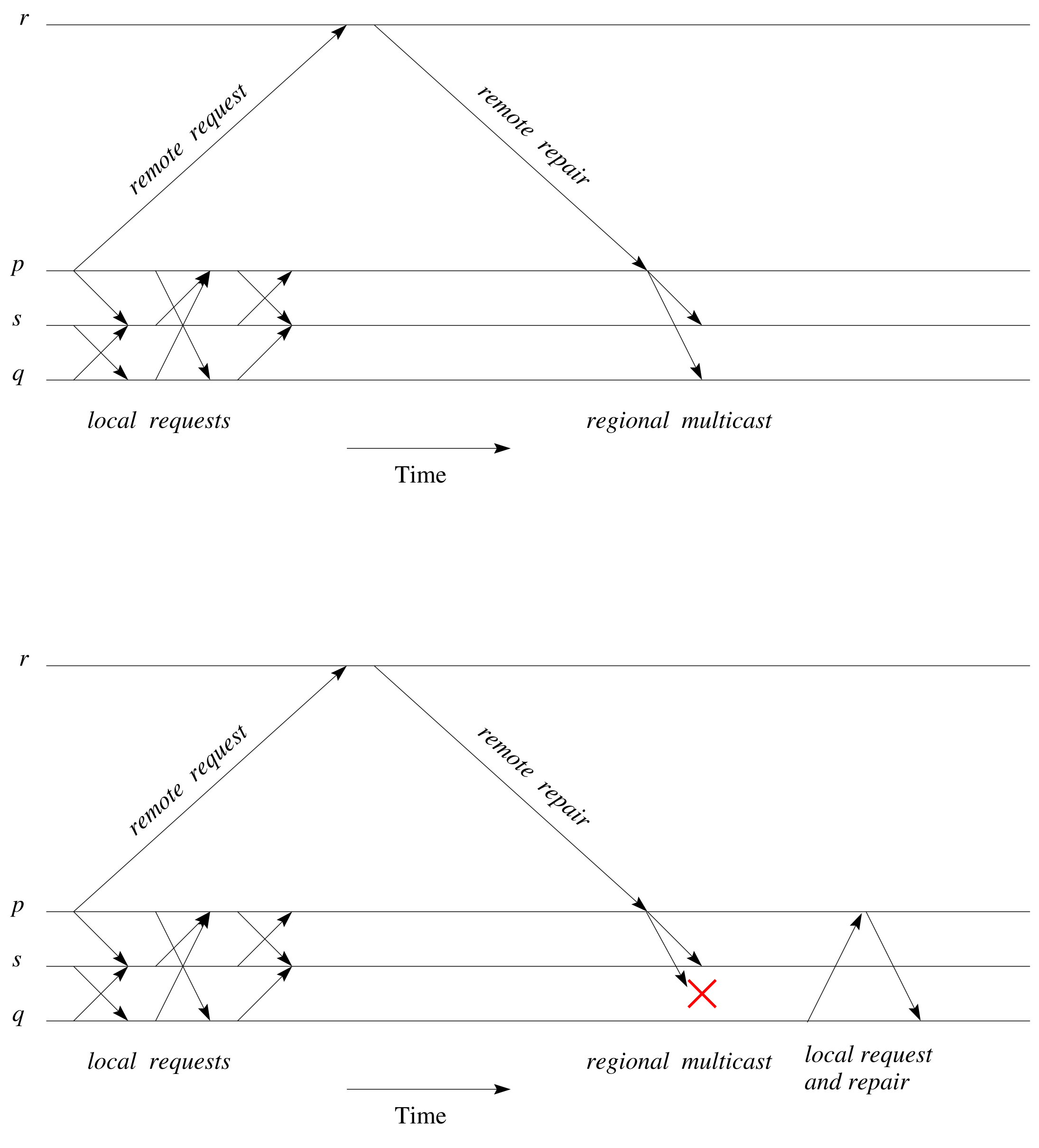


Figure 4.6: Optimization for reducing request traffic

again. If it hears a multicast for the same message from another member while it is waiting, it suppresses its own multicast. The waiting period is proportional to the propagation delay within its region. Although we introduce back-off delays before sending regional multicast, this delay is constant for any given λ and does not increase with the size of the region.

4.5 Performance Analysis

This section analyzes the performance of RRMP under several important metrics.

4.5.1 Implosion Avoidance and Robustness

RRMP avoids message implosion by distributing the responsibility of error recovery among all members in the group. If a member suffers from a high loss rate, its retransmission requests are sent to randomly selected neighbors rather than concentrated on a single repair server as in tree-based protocols. Robustness is also improved because the failure of a single member now has much reduced impact on other members in the group.

4.5.2 Recovery Latency

Recovery latency is defined as the interval between the time a loss is detected and the time it is repaired. In RRMP, a member missing a message tries to repair the loss simultaneously through local recovery and remote recovery. During the local recovery phase, a member sends requests to randomly selected members in its region. The recovery latency depends on how many members in the region have the message. In the worst case only a single member has the message. Epidemic theory shows that the expected time for the message to propagate to the entire region in this case is proportional to the log of the region size [Pit87, DGH+87]. Figure 4.7 shows the probability for a member to receive a repair in a particular request for a region of 30 members, with one member holding the message initially. The analysis used to compute this figure is similar to previous work on epidemic algorithms [Bai75, OD81, Pit87, DGH+87, BHO+99].

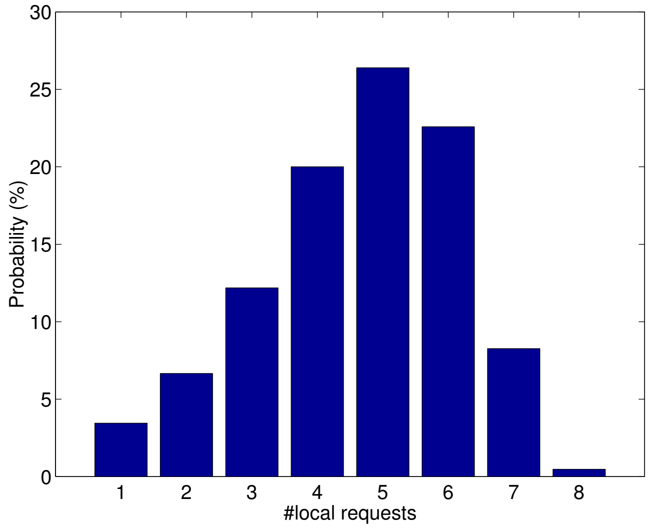


Figure 4.7: The probability for a member to receive a repair in a particular request for a region of 30 members, with one member holding the message initially.

Recovery latency can be reduced if a member sends requests to multiple destinations at a time, although this may increase the number of duplicate repairs. In addition, when a member selects a destination for its local request, it could bias its selection to favor members which have given good responses in the past. This, however, requires a member to maintain additional state for its neighbors. Alternatively, each member can advertise its smoothed loss rate periodically in its local session messages (described in Chapter 5). Due to differences in network conditions and local processing capacity, it is possible for some members to have a higher loss rate than others. Instead of selecting a destination uniformly at random from all other members in the region, a member can make its selection based on the loss rate of different members. In the future we plan to investigate these optimizations in detail.

During the remote recovery phase, a member sends a request to a remote member with probability $P = \lambda/n$, where n is the size of the local region. Assume that the entire region missed a message. The number of remote requests sent has a binomial distribution with parameters n and P. As $n \to \infty$, $P \to 0$ and $nP \to \lambda$. Hence for large regions the distribution can be approximated by a Poisson distribution with parameter λ [Dur94][5]. The probability that k requests are sent is $e^{-\lambda}\frac{\lambda^k}{k!}$. Figure 4.8 shows how the distribution changes with different values of λ. When $\lambda = 2$, for example, it is most likely that either one request or two requests will be sent to an upstream region. The choice of λ reflects a trade-off between recovery latency and repair duplication. As shown in Figure 4.8, when λ is small, there is a substantial risk that no remote request is sent due to randomization, leading to increased recovery latency. Fortunately, the probability of this happening is $e^{-\lambda}$, which decreases exponentially with λ as shown in Figure 4.9. When $\lambda = 4$, for example, the probability is only 1.8%. Hence increasing λ can reduce the expected error recovery latency and improve the robustness of the protocol against loss of request and repair messages. On the other hand, large λ increases the number of duplicate repairs as explained in the following subsection.

[5]A similar observation is made in the Search Party protocol [CM99], although the details are very different.

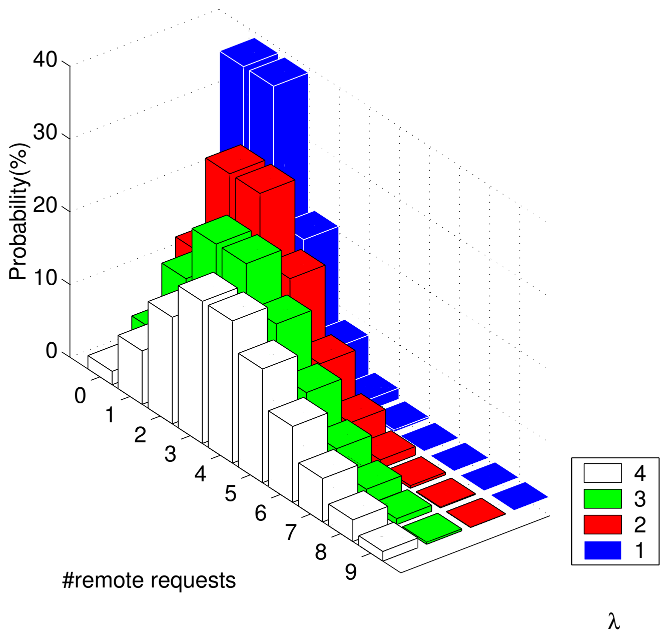


Figure 4.8: For large regions, the number of remote requests sent when all receivers missed a message approximately follows a Poisson distribution with parameter λ. The probability that k remote requests are sent is $e^{-\lambda}\frac{\lambda^k}{k!}$.

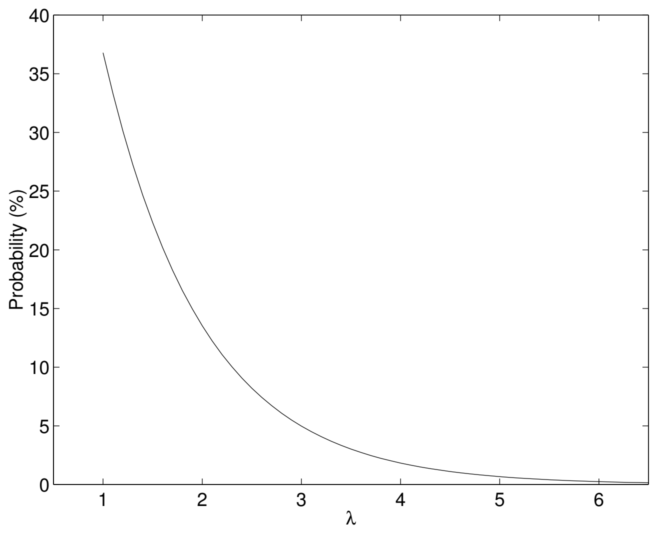


Figure 4.9: For large regions, the probability that no remote request is sent when all receivers missed a message decreases exponentially with λ.

The concurrent execution of the local recovery phase and the remote recovery phase increases the likelihood that a local message loss will be repaired by a local member. This is especially important when the latency between two regions is much higher than the latency within a region. For example, in Figure 4.1, the receivers at the left branch of the tree below the point of the message loss are likely to get retransmissions from the two receivers at the right branch, thus avoiding inter-region latency.

4.5.3 Repair Duplication

Duplicate repairs can be received for various reasons. For example, inaccurate RTT estimation may lead to premature timeout, which can trigger duplicate repairs. In addition, if multiple members in a region simultaneously receive repairs from upstream members for the same message, duplicate regional multicasts may be sent due to randomization. The concurrent execution of local recovery and remote recovery may also introduce multiple repairs for the same message: upon detection of a loss, a member sends a remote request with probability λ/n. If the lost message is later recovered locally, the repair from the remote member will become a duplicate. The number of duplicates in this case decreases with n but increases with λ. Hence λ controls a trade-off between recovery latency and repair duplication: large λ reduces recovery latency at the price of a higher number of duplicate repairs. On the other hand, small λ reduces the number of duplicate repairs but leads to longer recovery latency. Applications with different delay/bandwidth requirements can tune λ to suit their own needs.

4.5.4 Locality of Recovery

Locality of recovery can be measured by comparing the number of members receiving a repair with the number of members missing the message. Ideally, only those members which have missed a message are exposed to the repair traffic. In RRMP, a repair can be sent either in unicast or in regional multicast. If a repair is sent in unicast, it has perfect locality because a member will receive the message only if it has asked for it. On the other hand, if a repair is sent in regional multicast, its locality depends on the percentage of local members missing the message. Recall that a receiver executes the local recovery algorithm and the remote recovery algorithm concurrently upon detection of a loss. If the loss affects only a small portion of the region, a receiver is likely to recover the loss from a neighbor first. If it has also sent a remote request, the corresponding repair will be discarded as a duplicate upon receipt and will not trigger a regional multicast. In fact, if the ratio of inter-region latency to intra-region latency is sufficiently high (which is usually the case in a WAN), a receiver will always repair a loss from a neighbor first as long as at least one local member has the message. Consequently, a repair will be sent in a regional multicast only if the entire region missed the message, in which case it has perfect locality.

4.6 Conclusion

Error recovery is an essential part of a reliable multicast service. This chapter has presented a randomized reliable multicast protocol called RRMP which provides efficient error recovery in large multicast groups. Compared with traditional tree-based protocols, RRMP achieves better load balancing by diffusing the responsibility of error recovery among all members in the group and improves the robustness of the system against process failures. Error recovery latency is also improved through the concurrent execution of the local recovery phase and the remote recovery phase. In the following chapter, we describe how to construct the error recovery hierarchy in RRMP.

Chapter 5

Formation of The Error Recovery Hierarchy

This chapter describes an algorithm for constructing the error recovery hierarchy in RRMP. The algorithm groups receivers into local regions based on administrative domains and organizes different regions into a hierarchy according to their distance from the sender. This is achieved by periodic exchanges of session messages among all members in a group. We adopt an idea from the scalable session message protocol [SEFZ98] in SRM which divides session messages into two categories: *local* session messages and *global* session messages. Local session messages are multicasts restricted within a local region and global session messages are multicasts that reach the entire group. Session messages are also used to synchronize state among receivers and to help a receiver detect the loss of the last message in a burst, an idea previously used in the SRM protocol [FJM+95]. The global session interval T_{gs} and the local session interval T_{ls} are configuration parameters of the system.

This chapter is structured as follows. Section 5.1 describes how to group receivers into local regions. Section 5.2 describes how to organize different regions into an error recovery hierarchy. Properties of the hierarchy formation algorithm are discussed in Section 5.3. We will refine the definition of a *parent* region introduced in the previous chapter. Related work in this area is described in Section 5.4. In particular, we compare our protocol with the scalable session message protocol [SEFZ98] in SRM. Section 5.5 concludes.

5.1 Formation of Local Regions

RRMP divides receivers into local regions based on administrative domains and uses administrative scoping to restrict the scope of local session messages. For example, all receivers in Cornell University can form a local region. Receivers within a region periodically exchange local session messages to learn about their neighbors and to estimate the size of the region. A local session message from a receiver contains the largest sequence number it has received from the sender (for loss detection purposes) and some optional fields. For example, a receiver could advertise its smoothed loss rate in its local session message. Such information might be helpful during the local error recovery process because a receiver missing a message can toss a weighted coin to select destinations for its local requests based on the error probabilities of its neighbors as described in the previous chapter. Robustness of the protocol is achieved through soft-state timers: if a receiver has not been heard from for a certain period of time, it is assumed to have crashed or left the region.

5.2 Establishment of the Hierarchy

Once local regions are formed, a member needs to establish group membership knowledge about members in its parent region. To make this possible, every member periodically announces its presence by multicasting a global session message to the group. If all members send global session messages at a fixed interval, the total number of global session messages increases linearly with the size of the group. To reduce bandwidth consumption, we keep the expected number of global session messages from each region to a constant using a randomized algorithm similar to that used in the remote error recovery phase as described in Chapter 4. More specifically, during each session interval T_{gs}, a member r multicasts a global session message only with probability λ'/n , where λ' is a system configuration parameter and is not necessarily the same as the λ used for error recovery. For example, if a local region has 100 members and each member multicasts a global session message with probability 4/100, then on average 4 global session messages are sent to the group during each session.

The global session message from member r contains the largest sequence number r has received from the sender, its hop counts from the sender, the initial TTL value of the message, and some optional fields (e.g. its smoothed loss rate). A member obtains its hop counts from the sender through the TTL field of data or session messages it received from the sender. Let s denote the sender of the multicast group and $\texttt{Hop}(r, s)$ denote the number of hops between r and s. Figure 5.1 illustrates the format of a global session message. Both local and global session messages have constant size.

max seqno	Hop(r, s)	initial TTL

Figure 5.1: Format of global session message.

When a member p receives a global session message from a remote member r, p needs to decide whether it selects r as a member in its parent region. Recall that members in the sender's region have no parent region. If p is not in the sender's region, it makes its decision in two steps. First it checks whether r is an upstream member. If so, in the second step it checks whether r is close enough to be an appropriate destination for sending remote requests. In the following we describe the details of the two decision steps.

During the first step, p compares its position in the multicast tree with that of r to decide whether r is upstream in the hierarchy. Since the topological structure of the underlying tree is not explicitly available at the transport layer, p deduces their relative positions based on its distance from r and their distances from the sender. More specifically, p considers r to be an upstream member if the following two conditions hold:

- r is closer to the sender than p ;
- p is closer to r than to the sender.

Using the `Hop` notation defined previously, the above two conditions can be expressed as follows:

$$\text{Hop}(r, s) < \text{Hop}(p, s)$$

$$\text{Hop}(p, r) < \text{Hop}(p, s)$$

All distance information used in the comparison can either be calculated by p or is included in r's global session message. Figure 5.2 illustrates a situation where the positions of s, p, and r satisfy both conditions.

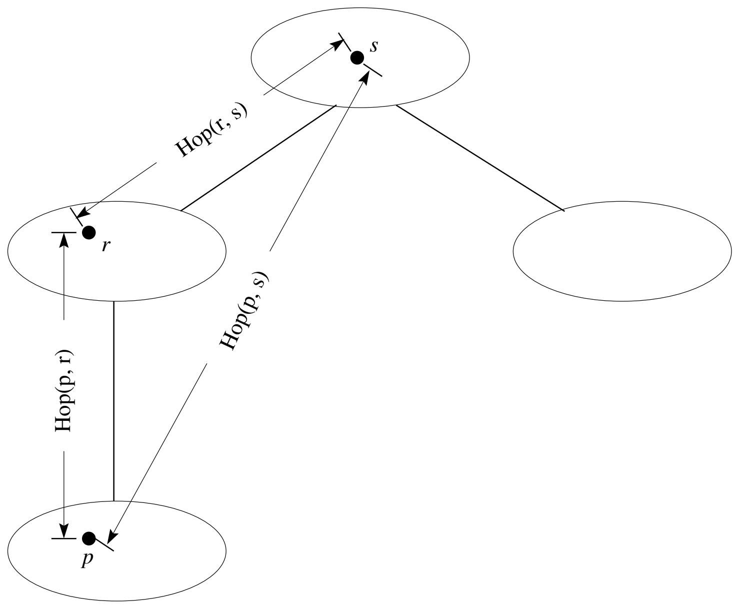


Figure 5.2: Example of upstreamness test. In this case, we have $\text{Hop}(r, s) < \text{Hop}(p, s)$ and $\text{Hop}(p, r) < \text{Hop}(p, s)$. Hence r is an upstream member of p.

It is helpful to understand why these two conditions are sensible measurements of *upstreamness*. Recall that members in the parent region are potential destinations for remote requests. If r is closer to the sender than p, it is likely the case that r can detect a message loss and recover from it earlier than p. This is checked in the first condition. In addition, if p is closer to r than to the sender, then it is faster for p to get a repair from r than from the sender. This is checked in the second condition. Both of them are important criteria in choosing appropriate destinations for remote requests. Consider the situation in Figure 5.3. In this case, r is still closer to the sender than p. However, because its position is at the other side of the multicast tree, it is inefficient for p to solicit a retransmission from r. Doing so not only increases error recovery latency but also wastes network bandwidth.

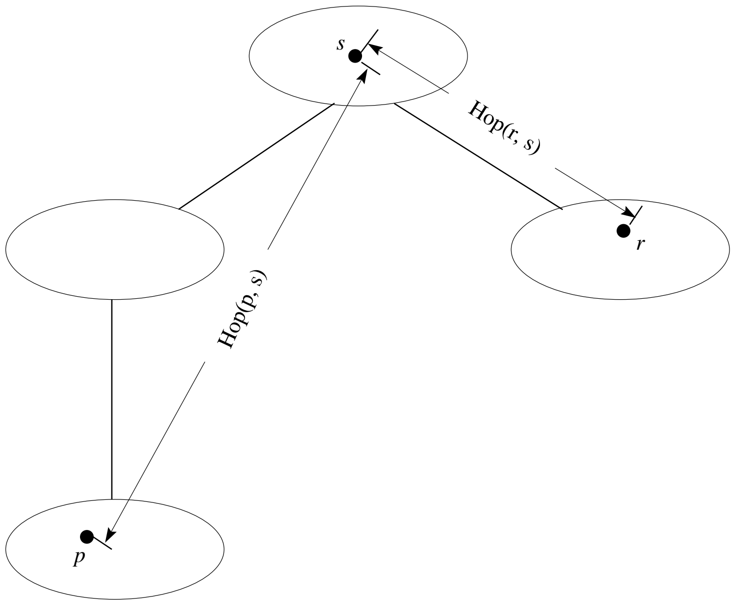


Figure 5.3: Another example of upstreamness test. In this case, we have $\text{Hop}(r, s) < \text{Hop}(p, s)$ but $\text{Hop}(p, r) > \text{Hop}(p, s)$. Hence r is *not* an upstream member of p.

If r is an upstream member, in the second step p compares its distance from r with that from other upstream members which it has heard from recently, and chooses a set of closest ones as members in its parent region. More specifically, p maintains its current estimate of least upstream members in a list. Each element in the list contains the following information for an upstream member r:

- r's network address ;
- p's hop counts from r ;
- a soft state timer.

This timer is reset whenever p hears from r. When it expires, r is dropped from the list. This does not imply that r has left the group because a member only sends a global session message with a certain probability. Hence other members in r's region may have been added to the list. The property of the list is that p's distance from the farthest member in the list is at most H hops more than its distance from the closest member. This is illustrated in Figure 5.4. Whenever p adds a new upstream member to the list, it checks whether the property still holds. If not, members which are too far away (either the newly added member or some existing ones) are dropped from the list. H is a system configuration parameter that controls the degree of heterogeneity in the parent region.

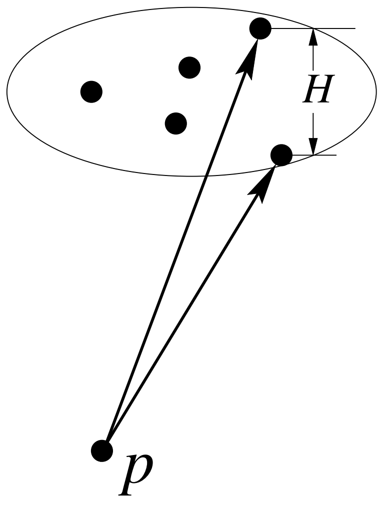


Figure 5.4: Member p selects a set of closest upstream members to be in its parent region. The algorithm requires that p's distance from the farthest member in the region is at most H hops more than its distance from the closest member. H controls the degree of heterogeneity in the parent region.

The choice of H reflects a trade-off between load balancing and error recovery latency. With large H, more members are likely to stay in the parent region. In the extreme case, a member may include all upstream members to be in its parent region. While this provides good load balancing for serving remote requests and a high degree of fault-tolerance against process failures, it also increases the possibility of soliciting retransmissions from upstream members which are quite far away. In contrast, with small H, a member is likely to send requests to nearby members, which reduces error recovery latency. In the extreme case, each member can keep only one upstream member in its parent list. This member, however, now bears the entire burden of error recovery for its downstream members.

If a member does not have any upstream member in its list, it chooses the sender as the default destination for its remote requests. This is the case when the member first joins the group. Figure 5.5 shows an example run of the algorithm described in this section. In this figure, p has two upstream regions, one on top of the other. Members s, q, and r are in the top-most region (s is the sender), while x, y, and z are in p's least upstream region. We use arrows to denote global session messages received by p even though these messages are actually sent to the entire group. For simplicity, we assume that H is larger than the distance within a region but smaller than the distance between the two regions. Initially p selects s as the default member in its list. When it receives a session message from r, it adds r into its list of least upstream members. Later y sends a global session message. Upon receipt of the message, p drops s and r from its list because they are now too far away. y becomes the only member in the list. Later p hears from x and adds it into its list. When p receives a global session message from q, it finds that q is too far away to be in its list and hence ignores the message. Finally, z is added into p's list.

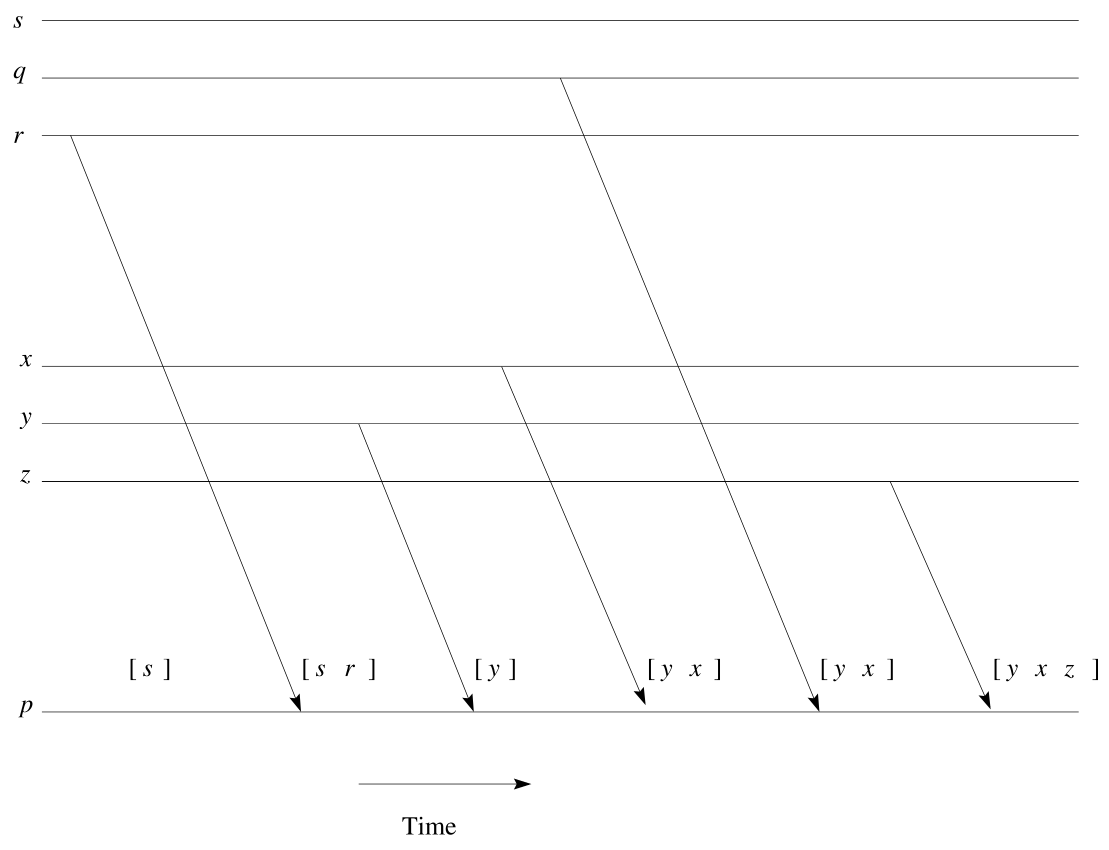


Figure 5.5: An example run of the algorithm to choose the parent region for member p .

5.3 Properties of the Algorithm

In our algorithm, the parent region for a receiver does not necessarily correspond to a local region. Because each receiver multicasts a global session message only with a certain probability and H may be smaller than the diameter of a local region, it is possible that a receiver's parent region contains only a subset of receivers in its least upstream region. Better load balancing can be achieved for serving remote requests if a receiver knows more members in its parent region, but the correctness of the protocol does not depend on the completeness of such knowledge. Nor does it require that such knowledge be consistent across different receivers in the same local region.

In addition, if a receiver has multiple upstream regions with similar distances, its parent region may contain a mixture of receivers from different local regions. Figure 5.6 illustrates a situation where region 4 has two upstream regions with similar distances. In this case a receiver in region 4 may select a mixture of receivers from region 2 and receivers from region 3 to be in its parent region. Since each receiver independently selects the destination for its remote request when an entire region misses a message, this scheme increases the possibility of getting a repair when some links in the network get congested. This is in contrast to a tree-based protocol in which all receivers in a local region rely on a single repair server for error recovery.

So far we have assumed that the algorithm uses hop counts to measure the distance between two receivers. The algorithm can be easily modified to use latency or round trip time as the distance metric. For example, currently the property of the parent region for receiver p requires that p's distance from the farthest member in

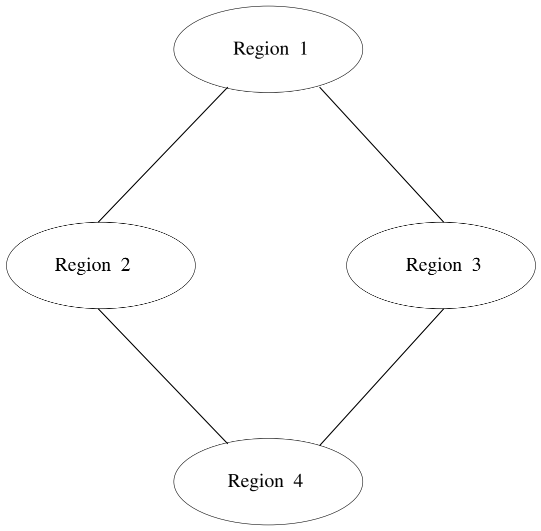


Figure 5.6: Formation of error recovery hierarchy in RRMP. The parent region of a receiver in region 4 may contain a mixture of receivers from region 2 and receivers from region 3.

the region is at most H hops more than its distance from the closest member. Should round trip time be used as the metric, the property would become: p's RTT from the farthest member in the region is at most T seconds larger than its RTT from the closest member. As discussed in the previous chapter, measuring RTT between two receivers on the Internet is a difficult problem due to its large variation. It is even more challenging to measure RTTs between many pairs of receivers when the clocks at these receivers are not synchronized. One optimization proposed in the previous chapter is to let all receivers in a local region share the round trip time estimation to a remote receiver. This is appropriate when the latency between two regions is much higher than the latency within a region, which is usually the case in a WAN. In addition, it is possible to consider a combination of several metrics such as hop counts, latency, loss rate, bandwidth, etc.. We leave this issue for future work.

Because of its randomized nature, RRMP is robust against transient inconsistency in group membership that can arise during process joins and leaves. It has higher memory requirements than tree-based protocols because each receiver needs to keep information about other receivers in its region as well as receivers in its parent region.[1]

5.4 Related Work

Related work has been done in tree-based protocols in the context of building an error recovery tree. The performance of tree-based protocols usually depends heavily on the quality of the tree. They can be classified as using either a *static* tree or a *dynamic* tree. The RMTP protocol is an example of using a static tree. In this protocol, specific machines are chosen to serve as repair servers and are statically organized into an error recovery tree. Every repair server periodically multicasts an advertisement message with a pre-specified TTL value. A receiver interested in joining the group can estimate its hop counts from different repair servers through the TTL field of advertisement messages it has received. Then it selects the closest one as its repair server. In effect, each repair server builds a local region in its proximity using TTL-based scoping.

TMTP is a protocol that uses a dynamic tree. In this protocol, repair servers are dynamically organized into a tree based on expanded ring search. A receiver always chooses the closest repair server as its parent, even if the server is downstream in the underlying multicast tree. In addition, several repair servers can form a loop of parent-child relations.

[1]In addition, a receiver in RRMP needs to buffer received messages for possible retransmissions. Buffer management is described in Chapter 6.

The Tracer protocol [LPGLA98] uses the MTRACE packet of IGMP protocol to discover the routing path from a receiver to the sender. In this protocol, every receiver sends an MTRACE query packet to the sender. As this query is passed hop-by-hop from the receiver to the sender, it records the path in the data portion of the packet. When the query reaches the sender, a response is returned to the receiver as a standard unicast packet. Based on the multicast path information, receivers are organized deterministically into a tree structure to achieve efficient local recovery. This approach, however, may put a heavy burden on the sender in a large multicast group because the sender needs to return a response for every receiver in the group [RM99a, RM99b].

Other related work includes the scalable session message protocol [SEFZ98] that proposes a self-configuring algorithm for establishing a hierarchical structure for distributing session messages. In this protocol, members are dynamically organized into a hierarchy of global members and local members. Global members send *global* session messages to the entire group, while local members send *local* session messages with a restricted scope. The hierarchy is established using a stochastic algorithm based on randomized timers and a set of *appropriateness* measures. For example, a global member with few local members in its region has a high appropriateness to switch to a local member if there is another global member nearby.

Both the RRMP protocol and the scalable session message protocol use a mixture of global session messages and local session messages to form a hierarchical structure. However, the two protocols are different in significant ways. RRMP is designed for "one-to-many" multicast applications. It forms local regions based on administrative domains and restricts the scope of local session messages using administrative scoping. Regions are organized into an error recovery hierarchy according to their distance from the sender. In contrast, SRM is designed for "many-to-many" multicast applications. Consequently, the hierarchy built in the scalable session message protocol is not organized with respect to a given source. There is no notion of *upstreamness* in its algorithm. The protocol uses TTL-based scoping to restrict the propagation of local session messages. This requires that the underlying routing protocols support the semantics of TTL-based scoping. Moreover, the hierarchy there is used only for distributing session messages in the SRM protocol and not for sending retransmission requests and replies.

Recently several proposals have been made on router-assisted reliable multicast as described in Chapter 2. For example, Search Party is a protocol that extends the existing IP multicast model by introducing a new forwarding service called *randomcast* at routers. In this protocol, when a receiver detects a message loss, it sends a request in a randomcast packet which is forwarded randomly inside a multicast distribution tree. The recipient of the request sends the repair message in a *directed multicast* that restricts its scope to the lossy subtree. A receiver missing a message keeps sending requests as a Poisson process until it receives a repair.

Both RRMP and Search Party use randomization to improve robustness. However, the two protocols differ in significant ways. RRMP works well within the existing IP multicast framework. It builds its error recovery hierarchy at the transport level without imposing any specific structure inside a region. In contrast, Search Party requires a new forwarding service from routers. It uses the underlying multicast tree itself for error recovery and avoids the need to construct a separate hierarchy. The forwarding service of randomcast relies on topological information of the multicast tree which is only available at the network level. The two protocols are also different in how request and repair messages are sent and have different performance characteristics.

5.5 Conclusion

Building an efficient error recovery hierarchy is essential to achieving high performance in reliable multicast protocols. This chapter has described the hierarchy formation algorithm in RRMP that organizes receivers in a multicast group into a hierarchy of local regions. Through periodic exchange of session messages among all receivers in the group, a receiver can establish group membership knowledge about receivers in its local region as well as receivers in its parent region. Such knowledge is required to perform randomized error recovery as described in Chapter 4.

Chapter 6

Buffer Management

Reliable multicast delivery requires that a multicast message be received by all receivers in the group. Hence certain or all members need to buffer messages for possible retransmissions. Previous work has demonstrated that relying solely on the sender for retransmissions leads to the message implosion problem [FJM$^+$95, PSLB97]. Consequently, several reliable multicast protocols adopt a distributed error recovery approach that allows certain or all members to retransmit packets lost by other members. For example, in the SRM protocol [FJM$^+$95], retransmissions are performed by all members in the group. In tree-based protocols like RMTP [PSLB97], LBRRM [HSC95], and TMTP [YGS95], members are grouped into local regions based on geographic proximity and a repair server is selected in each region to perform retransmissions.

If a member wants to perform retransmissions for other members, it needs to buffer received messages for some period of time. Determining which receivers should buffer a message and for how long turns out to be a difficult problem. A conservative approach is to have every member buffer a message until it has been received by all current members in the group. However, this is inefficient in a heterogeneous network where the delivery latency to different members could differ by orders of magnitude. Moreover, some reliable multicast protocols adopt the IP multicast group delivery model in which receivers can join or leave a multicast session without notifying other receivers. Consequently, no single receiver has complete membership information about the group.

Buffer management algorithms in existing reliable multicast protocols reflect widely different strategies for deciding which members should buffer messages and how long a message should be buffered. In some tree-based protocols, a repair server buffers all data packets it has received in the current multicast session. For example, the RMTP protocol was originally designed for multicast file transfer. In this protocol, a repair server buffers the entire file in a secondary storage. The approach is feasible only if the size of data transmitted in the current session has a reasonable limit. For long-lived sessions or settings where repair servers lack space, the amount of buffering could become impractically large.

The SRM protocol does not buffer packets at the transport level. Rather, the application regenerates packets if necessary based on the concept of Application Level Framing (ALF) [CT90]. This requires that the application be designed according to the ALF principle and be capable of reconstructing packets. Even so, buffer management at the application level remains a challenge.

Some reliable multicast protocols use a stability detection algorithm to detect when a message has been received by all members in the group and hence can be safely discarded [GR00]. This requires members in the group to exchange message history information periodically about the set of messages they have received. In addition, a failure detection algorithm is needed to provide current group membership information.

Previously we and others proposed a message buffering algorithm for reliable multicast protocols that reduces the amount of buffer requirement by buffering messages on only a small set of members [OvRBX99]. More specifically, we assume that each member has an approximation of the entire membership in the form of network addresses. The approximation needs not be accurate, but it should be of good enough quality that the probability of the group being logically *partitioned* into disconnected subgroups is negligible. Upon receiving a message, a member determines whether it should buffer the message using a hash function based on its network address and the identifier of the message[1]. If a member missed a message, it uses the same hash function to identify the set of members which should have buffered the message and requests a retransmission from one of them.

This algorithm makes no use of network topology information. Consequently, it suffers from a tendency to do error recovery over potentially high latency links in the network. Hence the protocol will have a scalability problem in genuinely large networks. Desired is an algorithm that selects receivers to buffer a message based on geographic locations of different receivers. Unfortunately, our previous algorithm cannot be easily modified to incorporate such information. The work described in this chapter was motivated by this observation.

In the following, we report our work on optimizing buffer requirements in RRMP. This work extends our previous work on buffer optimization by proposing an innova-

[1]A commonly used identifier is [source address, sequence number].

tive two-phase buffering algorithm that explicitly addresses the variances in delivery latency for large multicast groups. The algorithm reduces buffer requirements by adaptively allocating buffer space to messages most needed in the system and by spreading the load of buffering among all members in the group. Unlike stability detection protocols, the algorithm does not require periodic exchange of messages and has low traffic overhead.

The rest of the chapter is organized as follows. Section 6.1 describes the details of our buffer management algorithm. Section 6.2 evaluates its performance using simulation. Limitations of the algorithm are presented in Section 6.3. Section 6.4 concludes.

6.1 Optimizing Buffer Management

As described in Chapter 4, the RRMP protocol distributes the responsibility of error recovery among all members in a group. Hence every member needs to decide how long a message should be buffered for possible retransmissions. The problem is that this involves a trade-off with error recovery latency. If a member discards a message and later receives a retransmission request for that message, it would be unable to answer the request. Due to the randomized nature of our error recovery algorithm, this does not necessarily compromise the correctness of the protocol because another request will be sent to a randomly chosen member upon timeout. As long as some member still buffers the message, the loss can be recovered eventually. However, error recovery latency is increased because more requests were needed to repair the loss. The problem is even more complicated in a wide area network where the latency between two regions can be significantly higher than the latency within a region. Since a member can receive a request either from a local member or from a remote member, it is difficult to determine how long a message should be buffered for potential requests.

In order to reduce buffer requirements effectively while minimizing its impact on recovery latency, RRMP adopts an innovative two-phase buffering scheme: feedback-based short-term buffering and randomized long-term buffering. When a message is first introduced into the system, every member that receives the message buffers it for a short period of time in order to satisfy local retransmission requests. Later when the message has been received by almost all members in a region, only a small subset of members in this region continue to buffer the message. The rest of the section describes the details of our scheme.

6.1.1 Feedback-based Short-term Buffering

First we investigate how long a member should buffer a message for local retransmission requests. Since the outcome of the initial IP multicast for each message can be different, it is undesirable to buffer every message for the same amount of time. For example, if only a small fraction of members in a region have received a message during the initial IP multicast, these members should buffer the message for a long period of time in order to satisfy local requests from other members. In contrast, if almost all members have received the message during the initial multicast, then the message can be discarded quickly. Ideally, we want to allocate buffer space to messages most needed in the system.

In RRMP, the buffering time for a message is based on an estimation of how many members in the region have received the message. One way to estimate this information is to let all members periodically exchange message history information about the set of messages they have received, an idea previously used in some stability detection protocols [GR00]. Here we propose a different scheme in which a member estimates this information based on the history of retransmission requests it has received. Recall that in RRMP every member missing a message sends local requests to randomly selected members in its region. Hence the likelihood that a member receives a request increases with the number of members missing the message. More formally, let n be the size of a region and p be the percentage of members in this region missing a message. The probability that a member will *not* receive any request is:

$$(1 - \frac{1}{n-1})^{np}$$

As $n \to \infty$, this probability can be approximated by e^{-p}, which decreases exponentially with p. Consequently, if a member has not received any request after a sufficiently long period of time, it can conclude with high confidence that almost all members in the region have received the message. Based on this observation, we design a *feedback-based* scheme for short-term buffering: when a member receives a message, it buffers the message until no request for this message has been received for a time interval T. Such a message is called an *idle* message and T is called the *idle threshold*. The choice of T depends on the maximum round trip time within a region and the confidence interval. We call this a *feedback-based* scheme because a member uses the retransmission requests it received as feedback to estimate how many members in the region still miss the message. Unlike stability detection protocols, our scheme does not introduce extra traffic into the system.

6.1.2 Randomized Long-term Buffering

After a message has become idle, a member may decide to discard it. However, due to the randomized nature of the algorithm, it is possible that a message is still missing at some receivers but has become idle everywhere else. These unlucky receivers will not be able to recover the loss if all other members have decided to discard the message. Moreover, since inter-region latency can be much larger than intra-region latency, a member may receive a remote request from a downstream member asking for a message that has become idle at all members in the region.

RRMP addresses this problem by providing long-term buffering for an idle message at a small subset of receivers in each region. The set of long-term bufferers are chosen randomly from all members in a region. More specifically, when a member detects that a message has become idle, it makes a random choice to become a long-term bufferer with probability P. P is chosen so that the expected number of long-term bufferers in the region is a constant C. For a region with n members, probability theory shows that the number of long-term bufferers has a binomial distribution with parameters n and P [Dur94]. As $n \to \infty$, $P \to 0$ and $nP \to C$. Hence for large regions the distribution can be approximated by a Poisson distribution with parameter C. (In Chapter 4 we applied a similar technique to analyze the number of remote requests sent when an entire region missed a message.) The probability that k members buffer an idle message is $e^{-C}\frac{C^k}{k!}$. Figure 6.1 shows how the distribution changes with different values of C. The choice of C reflects a trade-off between buffer requirements and recovery latency. With large C more members buffer an idle message, and hence an unlucky receiver in the previous scenario will recover the loss faster. On the other hand, small C reduces buffer requirements but may lead to longer recovery latency. In particular, it is possible that an idle message is buffered nowhere due to randomization. The probability of this happening decreases exponentially with C as shown in Figure 6.2. When $C = 6$, for example, the probability is only 0.25%.[2]

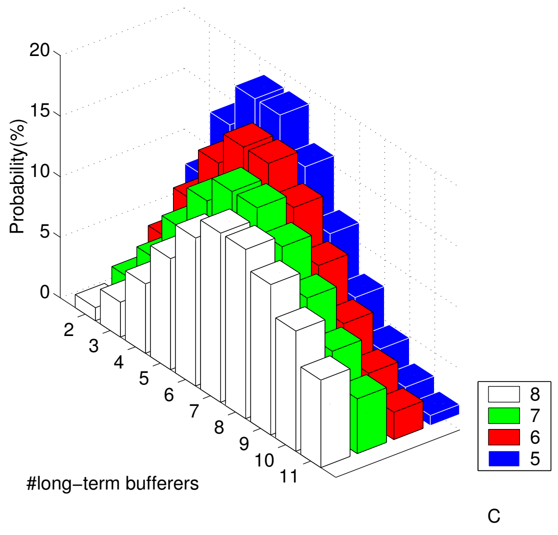


Figure 6.1: For large regions, the number of long-term bufferers for an idle message approximately follows a Poisson distribution with parameter C.

When the sender multicasts a stream of messages, the load of long-term buffering is spread evenly among all members in a region. This is in contrast to some tree-based protocols where a repair server bears the entire burden of buffering messages for a local region. Eventually even a long-term bufferer may decide to discard an idle message if the message has not been used for such a long time that it is highly unlikely any member may still need it.

[2]This is the probability that no receiver in a region buffers a message in its long-term buffer. It is *not* the probability that a receiver will miss a message. For example, if the receiver gets the message during the initial multicast, it will not need any error recovery at all.

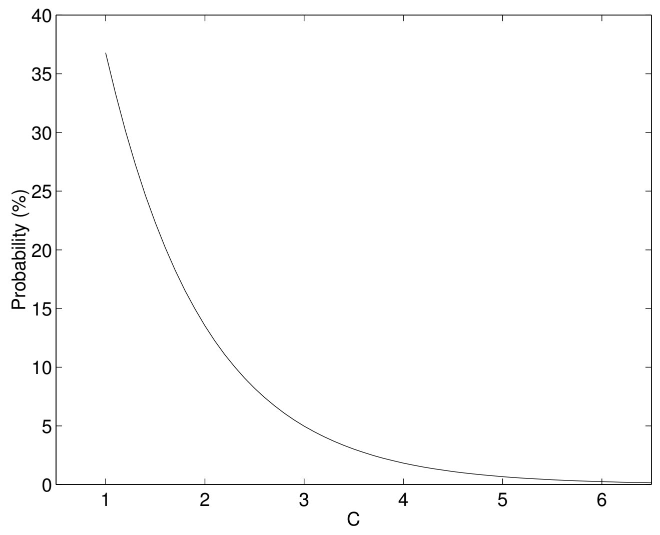


Figure 6.2: For large regions, the probability that no member buffers an idle message decreases exponentially with C.

Receivers may join or leave a multicast session dynamically. When a receiver voluntarily leaves the group, it transfers each message in its long-term buffer to a randomly selected receiver in the region. This avoids the situation where all long-term bufferers decide to leave the group, making a message loss unrecoverable.

6.1.3 Search for Bufferers

When a member p receives a remote request from a downstream member r for a message, there are three possibilities:

- p has received the message and still buffers it.
- p has never received the message.
- p received the message but has discarded it.

In the first case, p can immediately send the message to r. In the second case, p records r's request. Later when p receives the message, it will forward the message to r as described in Chapter 4. In the third case, however, p needs to search for a member which buffers the message.

One solution is for p to multicast r's request in its region. If a member has the message in its buffer, it multicasts a reply "I have the message" and then forwards the message to r. A randomized back-off scheme is used to suppress duplicate responses when multiple members buffer the message: upon receiving a request, a member waits a random amount of time before multicasting its reply in the region. If it hears a multicast for the same message, it suppresses its own multicast. The problem is how to choose an appropriate back-off period. As described earlier, the expected number of long-term bufferers for an idle message is C. Hence it is tempting to set the back-off period to be proportional to C. In practice, however, we have found that this approach occasionally leads to message implosion. Recall that in our feedback-based buffering scheme each member independently decides when a message has become idle based on retransmission requests it received from other members. Because of randomization, it is possible that a message has become idle and been discarded at one member but is still being buffered at many other members (i.e. the message has *not* become idle at all members in the region). If a multicast request is sent in this case, the back-off period will be too short to suppress duplicate responses effectively.

In order to avoid storms of multicast replies, RRMP adopts a different approach where a member conducts a random search in its region to find out a bufferer of the message. More specifically, when p receives r's request, it randomly selects a member q in its region and forwards r's request to q. p also sets a timer according to its estimated round trip time to q. Upon receiving r's request, q checks whether the message is still in its buffer. If so, it sends the message to r and multicasts a reply "I have the message" in its region. This reply notifies other members that the search process is over. If q has discarded the message as well, it joins p in the search process and tries to find a bufferer of the message[3]. If p does not hear a reply when its timer expires, it randomly selects another receiver in its region and repeats the above process. As time goes by, more and more members will join the search process. As long as at least one member in the region still buffers the message, r will receive the message eventually.

Figure 6.3 illustrates the search process in a region with four members, one of which is a bufferer. The horizontal direction represents different members in the group, and the vertical direction represents the amount of time that has elapsed since the search starts. We assume that the latency between any two members in the region is 5ms. Suppose member p_1 receives a remote request at time 0. It forwards the request to a randomly selected member p_2. Since p_2 does not have the message either, it forwards the request to p_3. After 10ms p_1 times out and sends another request to p_4, which is the bufferer. Upon receipt of the request, p_4 sends the message to the remote member and multicasts a reply in the region.

The search time for a message depends on the number of members that buffer the message. If the message has become idle at all members in the region, the expected number of bufferers is C. Hence increasing C can reduce search time at the expense of higher memory requirements. In particular, the search process is avoided if r's request arrives at a bufferer of the message.

[3]If q has never received the message, it will send retransmission requests as described in Chapter 4.

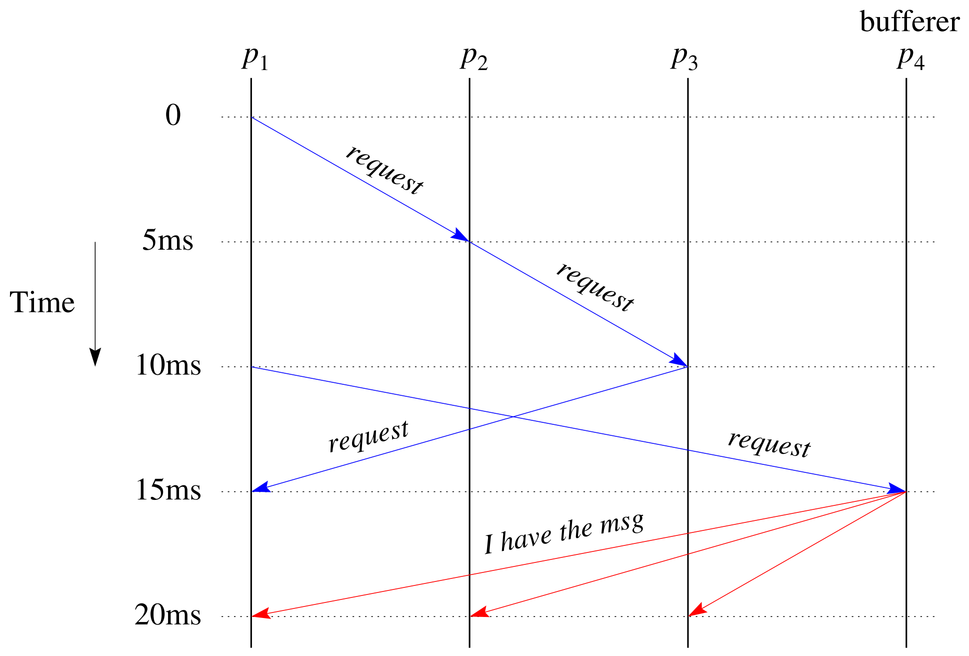


Figure 6.3: Search for bufferers in a local region

The above discussion is simplified in assuming that p is the only member receiving a remote request. As described in Chapter 4, when an entire region missed a message, on average λ members will send remote requests to an upstream region. As soon as one of them receives a remote repair, it will multicast the repair in its region.

6.1.4 Comparison with a Hash-based Scheme

In the RRMP protocol, the set of long-term bufferers are chosen randomly from all receivers in a region. Previously we and others proposed a deterministic algorithm [OvRBX99] that chooses a subset of receivers in a group to serve as bufferers using a hash function as described at the beginning of the chapter. It is interesting to compare these two approaches.

We believe that the choice reflects a trade-off between network traffic and computation overhead. Under the deterministic algorithm, a receiver can find out the set of bufferers for a message by applying the hash function to the network address of each member in its region. This avoids the latency and network traffic associated with the search process. However, it incurs certain computation overhead because the hash function needs to be calculated each time a message is received. In [OvRBX99] van Renesse proposed the design of an efficient hash function.

One advantage of the randomized algorithm is that it allows easy adaptation to group membership dynamics: when a receiver voluntarily leaves the group, it can transfer messages in its long-term buffer to randomly selected receivers in its region. It is not clear how this can be done easily with a deterministic algorithm.

6.2 Simulation Results

In this section, we evaluate the performance of our buffer management scheme using simulation. We focus on the behavior of the protocol in a local region. The round trip time between any two members in the region is 10ms. The idle threshold T is set to 40ms (i.e., 4 times the maximum round trip time). We assume that retransmission requests and repairs are not lost.

We first evaluate the effectiveness of our feedback-based short-term buffering scheme in a region with 100 members. We simulate the outcome of an IP multicast by randomly selecting a subset of members to hold a message initially. All other members simultaneously detect the loss and start sending local requests. We measure how long these initial members buffer the message. The result is shown in Figure 6.4

(note that the x-axis is in logarithmic scale). As can be seen from the figure, the amount of buffering time decreases as the initial IP multicast has reached more members.

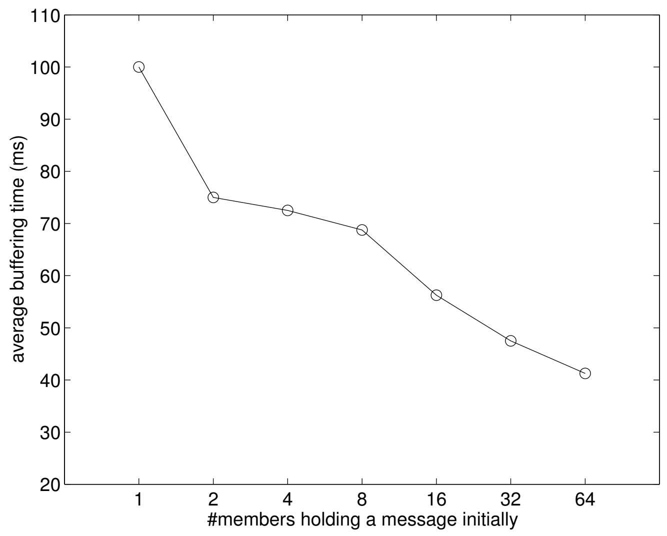


Figure 6.4: Effectiveness of feedback-based buffering. The x-axis is in logarithmic scale. The figure indicates that the amount of buffering time decreases as the initial IP multicast has reached more members.

In Figure 6.5 we take a closer look at one of the data points in Figure 6.4 where one member holds a message initially. We compare the number of members which have received the message with the number of members which buffer the message as error recovery proceeds. As can be seen from the figure, when only a small percentage of members have received the message, almost all of them buffer the message. The number of short-term bufferers declines rapidly when an overwhelming majority of members (96% in this case) have received the message. The results in these two figures demonstrate that our feedback-based scheme is effective in allocating buffer space to messages most needed in the system.

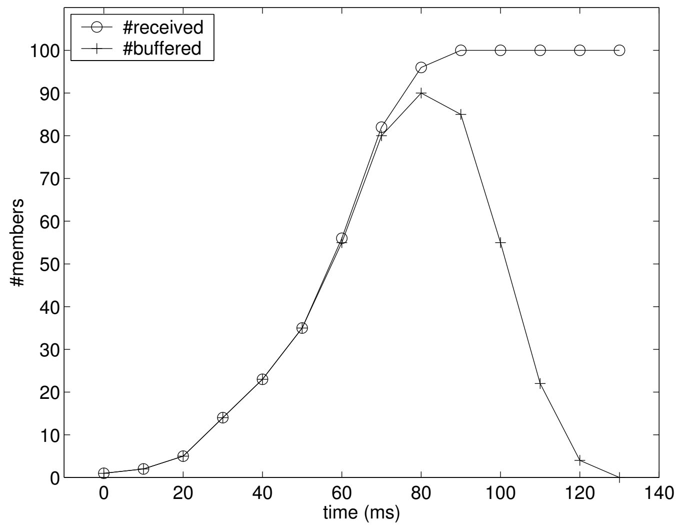


Figure 6.5: Comparison between the number of members which have received a message and the number of members which buffer the message as error recovery proceeds. The figure indicates that the number of short-term bufferers declines rapidly when an overwhelming majority of members have received the message.

Next we investigate the penalty in error recovery latency due to a need to search for a bufferer. We assume that a remote request arrives at a randomly chosen member in a region with 100 members. The simulation is repeated 100 times with different random seeds and the average is taken. Figure 6.6 shows that the search time decreases as the number of bufferers increases[4]. With 10 bufferers, for example, the average search time is 20ms (i.e. twice the round trip time). In a wide area network, the latency between two regions is usually much higher than the latency within a region. Hence the search time is likely to be a small fraction of the total recovery latency.

In Figure 6.7 we show how the search time changes when the size of the region increases from 100 members to 1000 members. The number of bufferers is fixed at 10.

[4]The search time is 0 if the request arrives at a bufferer.

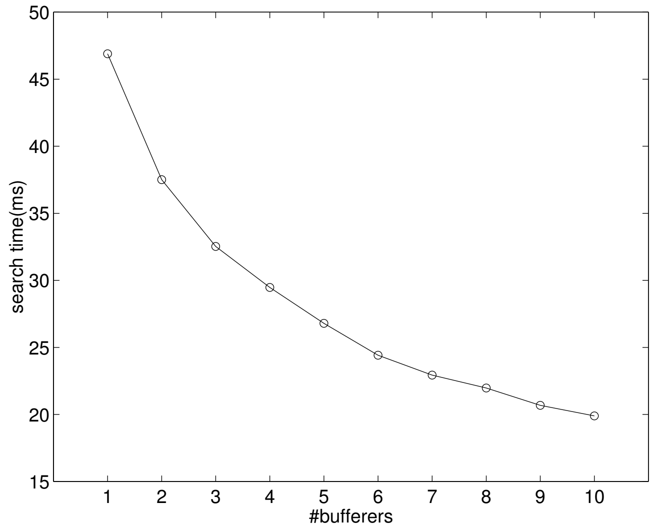


Figure 6.6: Search time decreases as the number of bufferers increases.

The figure indicates that the degree of increase in search time is much smaller than that in region size: when the region size increases by a factor of 10, the corresponding search time only increases by a factor of 2.2. With 1000 members, the percentage of bufferers is only 1%. Compared with the case where every member buffers the message, our algorithm reduces the amount of buffer space by a factor of 100.

6.3 Limitation

In RRMP, a member may discard a message before the message has been received by all members in the group. This is in contrast to stability detection protocols where a message is discarded only after it has been delivered everywhere. Consequently, our buffering scheme introduces a small probability of violating the reliability guarantee of the multicast service. Such probability can be made arbitrarily small with carefully

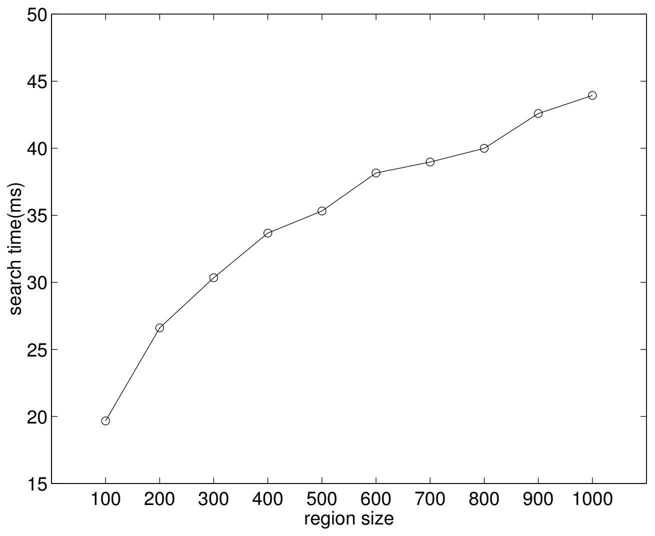


Figure 6.7: Search time as the size of the region increases.

chosen parameters for the protocol, but still must be accounted for when designing an application.

Applications that require stronger guarantees should use a protocol that provides better reliability, such as virtual synchrony [Bir97]. The probabilistic guarantees offered by RRMP have the benefit of superior scalability and intrinsic robustness in networks subject to message loss and process failures, but are not appropriate when absolute guarantees of reliability are needed.

6.4 Conclusion

Designing an efficient buffer management algorithm is challenging in large multicast groups where no member has complete group membership information and the delivery latency to different members could differ by orders of magnitude. This chapter has presented an innovative two-phase buffering algorithm that explicitly addresses variations in delivery latency seen in large multicast groups. Unlike tree-based protocols where a repair server bears the entire burden of buffering messages for a local region, RRMP achieves better load balancing by spreading the load of buffering among all members in the region. Compared with stability detection protocols, our buffering algorithm has low traffic overhead because it does not require periodic exchange of message history information among members in the group. Simulation results demonstrate that the algorithm has good performance.

Although we present our buffer optimization in the framework of RRMP protocol, similar techniques can be applied to other reliable multicast protocols as well. In the following we summarize the main ideas in our algorithm and discuss how they can be applied to the SRM protocol:

- The dissemination status of the initial IP multicast for each message can be different. A good buffering algorithm should adaptively allocate buffer space to messages most needed in the system.

- Retransmission requests can be used as feedback to estimate the dissemination status of a multicast message. In the SRM protocol, retransmission requests and replies are multicast to the entire group. If a receiver has not received any request for a message after a sufficiently long period of time, it can conclude that the message is stable. Such information can be helpful to the application in managing its buffer space.

- In a large multicast group, it may take a long time for a message to become stable. While research on stability detection focuses on optimizing buffer space after a message has become stable, our work aims to reduce buffer space even before stability has been achieved. In the context of SRM, instead of having every receiver buffer a message until the message is stable, a randomly selected subset of receivers can serve as bufferers for the message.

- In a wide area network, the latency between two regions can be much higher than the latency within a region. Our buffering algorithm addresses this difference in latency by dividing buffer space into two parts: the short-term buffer allows a local loss to be recovered quickly within the local region, while the long-term buffer serves to satisfy remote requests from downstream regions without consuming too much buffer space. Although the original SRM protocol is unstructured, extensions have been made to introduce an error recovery hierarchy into the protocol [LESZ98, SEFZ98]. Our two-phase buffering scheme can be applied in such a hierarchy.

Chapter 7

Simulation Results

In this chapter, we evaluate the performance of RRMP using the ns2 simulator [UCB]. As a target for comparison, we also implemented a tree-based reliable multicast protocol called TRMP in the simulator. In TRMP, a receiver missing a message gets a retransmission from its repair server in unicast. If the repair server itself has missed a message, it gets a retransmission from its least upstream server in the hierarchy and then multicasts the retransmission in its local region. The TRMP protocol is used to illustrate the problem of load concentration on repair servers and to investigate the performance penalty in the RRMP protocol due to randomization. It would be interesting to also compare RRMP with some existing tree-based protocols like RMTP. In fact, TRMP is based on RMTP and we believe that most of the results presented in this chapter can be applied to the RMTP protocol.[1] Unfortunately, the source code for the RMTP protocol was not made available to us by the developers, making a direct comparison impossible. The configuration parameters for the RRMP

[1]TRMP does not include all features of RMTP. In particular, it does not implement the flow and congestion control algorithms in RMTP.

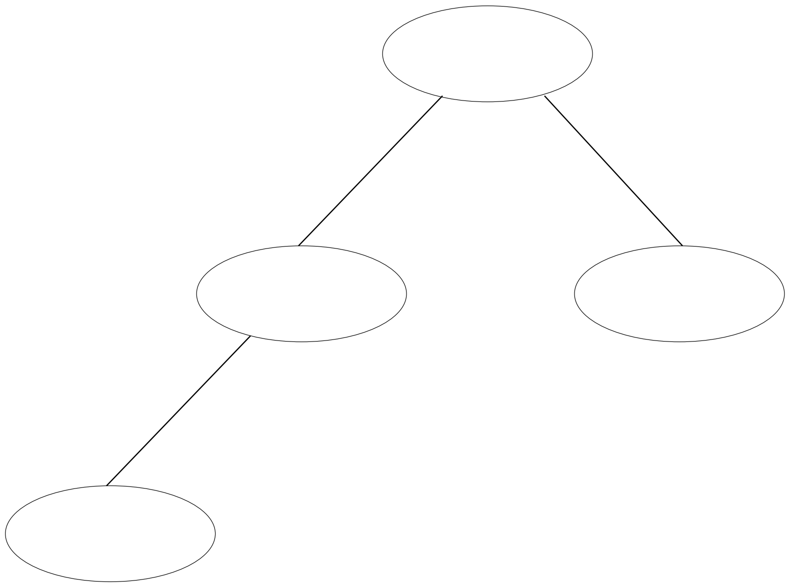
protocol are shown in Table 7.1. For simplicity, we assume that every receiver buffers received messages for a sufficiently long period of time.

Table 7.1: Configuration parameters for RRMP

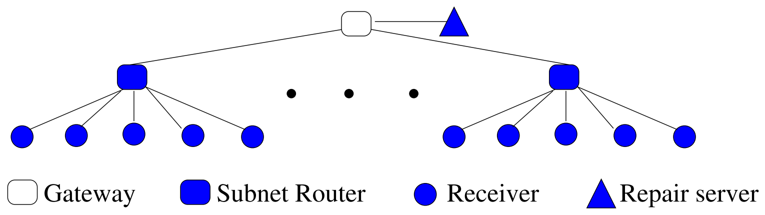
parameter	description	value
T_{maxl}	maximum interval between any two local requests	1 second
T_{maxr}	maximum interval between any two remote requests	5 seconds
T_{gs}	global session interval	1 second
T_{ls}	local session interval	1 second
H	degree of heterogeneity in a parent region	4 hops
λ'	expected number of global session messages per region	2

7.1 Test Description

We conduct two sets of simulation tests with different topologies and loss patterns. In the first set of simulations, the topology consists of 4 regions connected in a hierarchy as shown in Figure 7.1 (a). Each region consists of a number of subnets connected to a common gateway. Each subnet is modelled as a star topology with 5 machines connected to a subnet router. For the tree-based protocol, each region has a repair server connected to the gateway of that region. Figure 7.1 (b) illustrates the internal topology of a region. Two regions are connected if there is a link connecting their gateways. Such a link is called an inter-region link. It has a bandwidth of $1M$bps and a propagation delay of 50ms. In contrast, a link connecting two nodes within the same region is called an intra-region link. It has a bandwidth of $10M$bps and a propagation delay of 1ms. The sender is at the root of the hierarchy.



(a) Global topology



(b) Internal topology

Figure 7.1: Topology used in the first set of simulations

We introduce message loss by assigning a uniform loss probability of 5% on every inter-region link and 0.5% on every intra-region link. All messages are subject to loss, including retransmission requests and repairs. However, no message is lost on any link connecting a gateway with a repair server. Consequently, a repair server receives any message that is received by at least one member in its region. This is the optimal case for a tree-based protocol. Table 7.2 shows the link characteristics in the first set of simulations.

Table 7.2: Link characteristics in the first set of simulations

type	bandwidth	delay	loss rate
inter-region link	$1M$bps	50ms	5%
intra-region link	$10M$bps	1ms	0.5%

The topologies used in the second set of simulations are transit-stub networks generated using the GT-ITM network generator [CDZ97]. Links within transit domains are set to a bandwidth of $45M$bps to simulate multicast backbones. Links within stub domains have a bandwidth of $10M$bps. Links connecting transit domains to stub domains have a bandwidth of $8M$bps. Each direction of a link has a queue limit of 16 packets. All receivers are in stub domains, including the sender. Each stub domain is a local region. For the tree-based protocol, each stub domain also has a repair server connected to the root of that domain.

We introduce background traffic by establishing TCP connections between randomly selected nodes in the network. For each TCP connection, an FTP application is set up to transfer a file with infinite size. The background traffic caused observed loss rates between 0.71% and 8.02% on links connecting transit domains to stub domains, with a median loss rate of 4.29%. The loss rates for links within stub domains vary from 0% to 1.29%, typically around 0.32%. No message loss is observed on backbone links. In order to be fair to tree-based protocols, no background traffic is introduced on any link connecting a repair server with the root node of its region. Again, this is the optimal case for tree-based protocols. Table 7.3 shows the link characteristics in the second set of simulations.

Table 7.3: Link characteristics in the second set of simulations

type	bandwidth	loss rate
transit-transit link	$45M$bps	0%
transit-stub link	$8M$bps	0.71% – 8.02%
stub-stub link	$10M$bps	0% – 1.29%

In both sets of simulations, members exchange session messages to form the error recovery hierarchy as described in Chapter 5. Each simulation starts with a bootstrap period of 30 seconds during which the sender multicasts a global session message every second to let each receiver measure its hop counts from the sender. After the bootstrap period, the sender starts sending $1K$ byte data messages at a constant rate of 50 messages per second. The sender keeps sending data messages for 10 minutes during each simulation run. A total of 30000 messages are received at each receiver. Messages are delivered to the application in FIFO order.

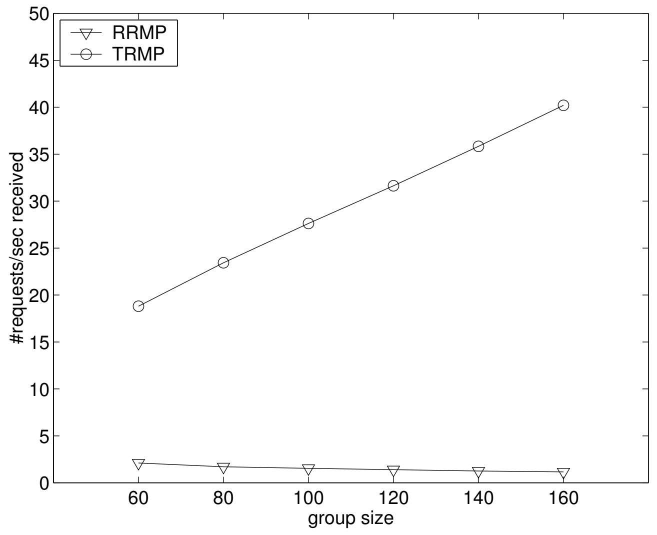
Both RRMP and TRMP use a mixture of unicast and regional multicast for repair messages. In the simulation, each local region is assumed to be in a different administrative domain and administrative scoping is used to restrict the scope of a regional multicast.

7.2 Load Balance

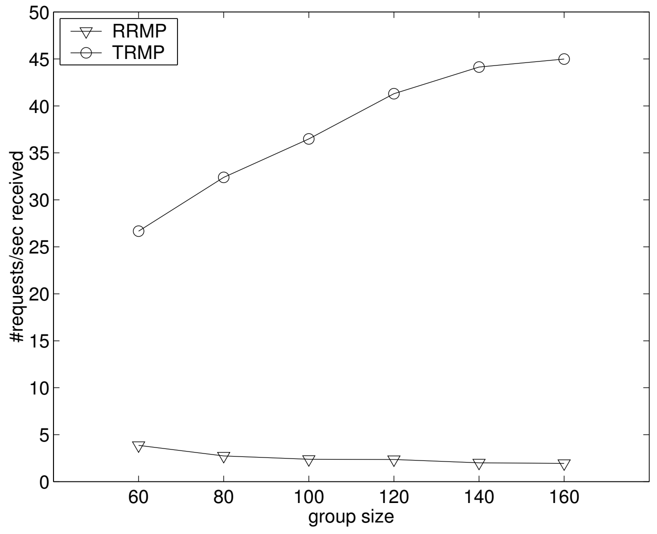
First we compare the load of request and repair traffic between the two protocols. The results are shown in Figure 7.2 and 7.3. Figure 7.2 compares the number of request messages received by a repair server in the TRMP protocol with the maximum number of request messages received by a member in the RRMP protocol during the simulation. Figure 7.3 compares the number of repair messages sent by a repair server in the TRMP protocol with that sent by the worst-case member in the RRMP protocol. The parameter λ for RRMP is set to 4. As can be seen from the figures, for both sets of simulations, the load on the repair server increases linearly with the group size for TRMP. This is because the repair server bears the entire burden of error recovery for its region. In contrast, the load for RRMP decreases slightly with the group size. This is because the probability that a member receives a remote request decreases with its region size, for any given λ.

One way to reduce the load on a repair server is to split a large region into several small ones. This is effective if all members in the region have roughly the same loss rate. Otherwise a single member suffering a high loss rate can put a heavy burden on its repair server even after the split. This is shown in Figure 7.4 for a group of 160 members when the loss rate of one receiver is increased from 1% to 28%.[2] We compare the number of repair messages sent to this lossy receiver by its repair server in TRMP with that sent by the worst-case member in RRMP. (The figure for request load is similar and hence omitted.) As can be seen from the figure, a lossy receiver can have a significant impact on its repair server in TRMP but only a limited impact on its neighbors in RRMP.

[2]This is in addition to any congestion loss caused by background traffic.

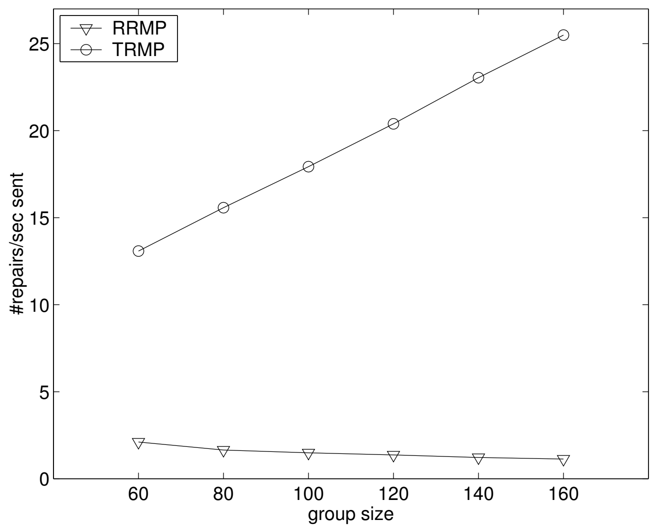


Test I: Tree topology, random loss

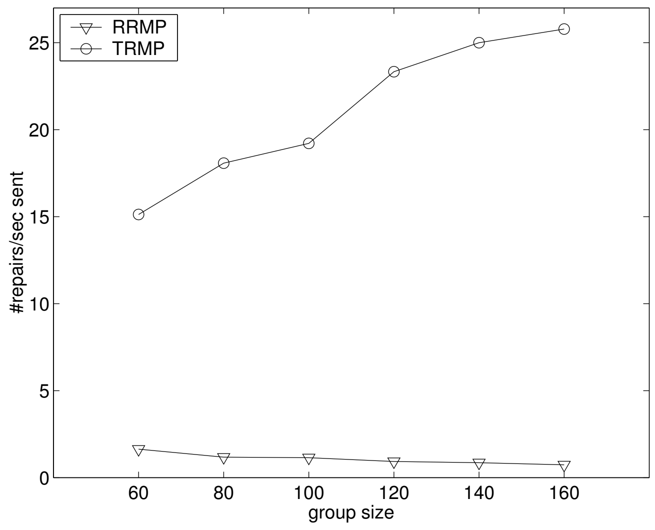


Test II: Transit-stub topology, congestion loss

Figure 7.2: Comparison of request traffic received by a repair server in TRMP with that received by the worst-case member in RRMP when the group size increases.

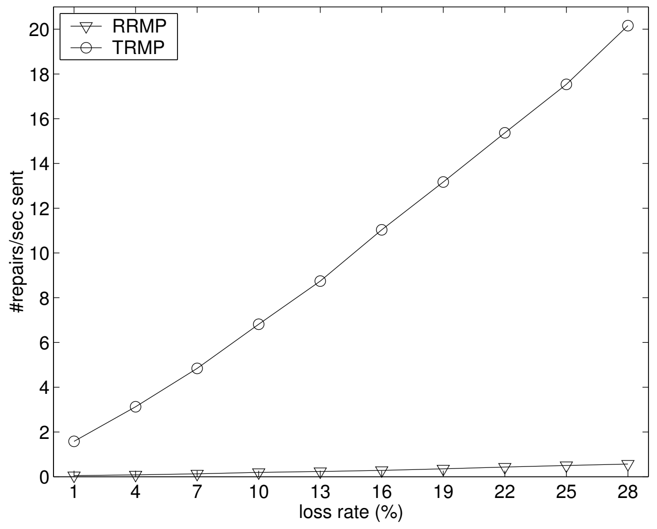


Test I: Tree topology, random loss

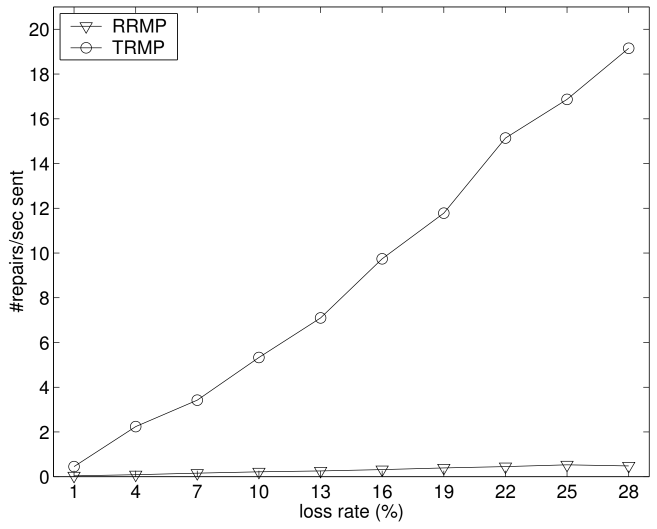


Test II: Transit-stub topology, congestion loss

Figure 7.3: Comparison of repair traffic sent by a repair server in TRMP with that sent by the worst-case member in RRMP when the group size increases.



Test I: Tree topology, random loss



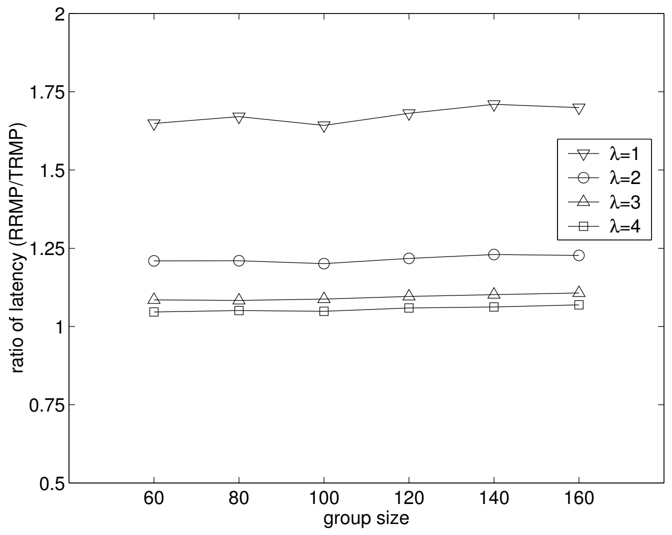
Test II: Transit-stub topology, congestion loss

Figure 7.4: Comparison of repair traffic sent to a lossy receiver by a repair server in TRMP with that sent by the worst-case member in RRMP.

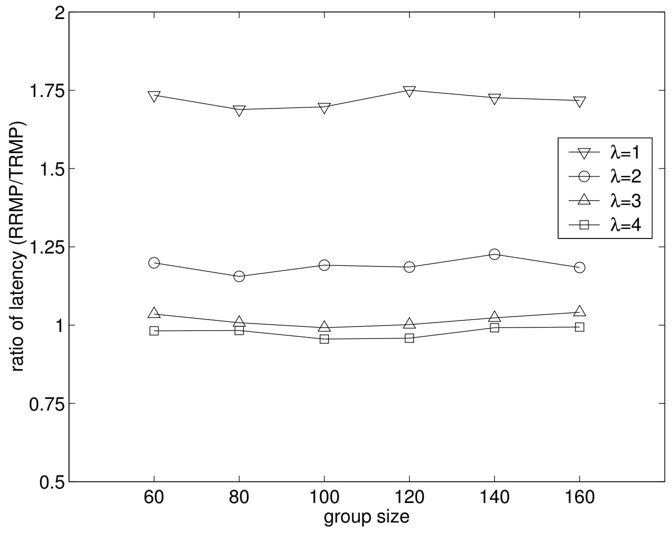
Another way to reduce the load on a repair server is to use a mixture of unicast and regional multicast for retransmissions. For example, in the RMTP protocol, a repair server multicasts a repair message if it has received several requests for that message. Again, this approach is ineffective if a single member suffers from a high loss rate.

7.3 Recovery Latency

Recovery latency is defined as the interval between the time a loss is detected and the time it is repaired. Each member measures the average recovery latency over all message loss it experienced during the simulations. We compute the ratio of recovery latency in RRMP to that in TRMP memberwise. Figure 7.5 shows the average ratio for different values of λ when the group size increases. As can be seen from the figure, there is an observable performance penalty for RRMP due to randomization when $\lambda = 1$ or 2. This is because for small λ there is a substantial risk that no remote request is sent when an entire region missed a message, leading to increased recovery latency. The analysis in Chapter 4 indicates that the probability of this happening decreases exponentially with λ. This is confirmed in Figure 7.5, which shows that the latency of RRMP improves when λ increases. When $\lambda = 4$, the latency of RRMP is almost as good as that of TRMP. In fact, in the second set of simulations the latency of RRMP is slightly better than that of TRMP. This is because sending multiple remote requests improves robustness against loss of request and repair messages. For any given λ, the latency of RRMP does not increase with group size, which indicates that the algorithm scales well.



Test I: Tree topology, random loss



Test II: Transit-stub topology, congestion loss

Figure 7.5: Error recovery latency

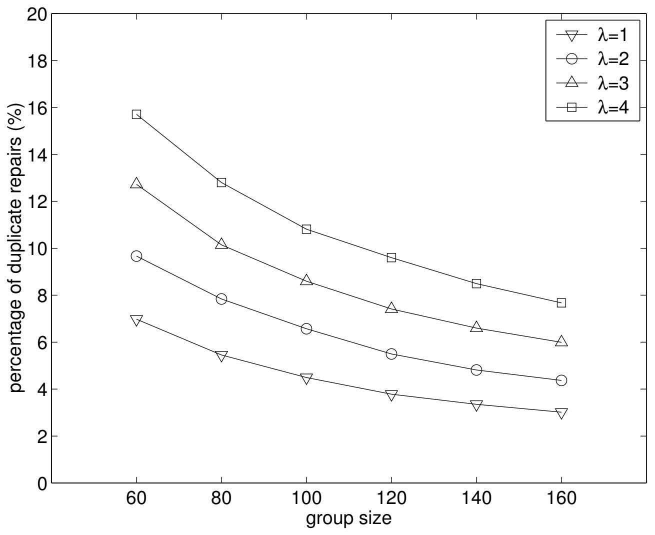
7.4 Repair Duplication

In RRMP, each member calculates the percentage of repair messages it has received that are duplicates. The result is averaged over all receivers in the group. Figure 7.6 shows that the percentage of duplication is low and decreases with group size, for any given λ. As described in Chapter 4, duplicate repairs can be generated due to (among other reasons) the concurrent execution of local recovery and remote recovery: upon detection of a loss, a receiver sends a remote request with probability λ/n. If the lost message is later recovered locally, the repair from the remote member will become a duplicate. The number of duplicates in this case decreases with n but increases with λ. Hence the protocol has better performance for large regions.

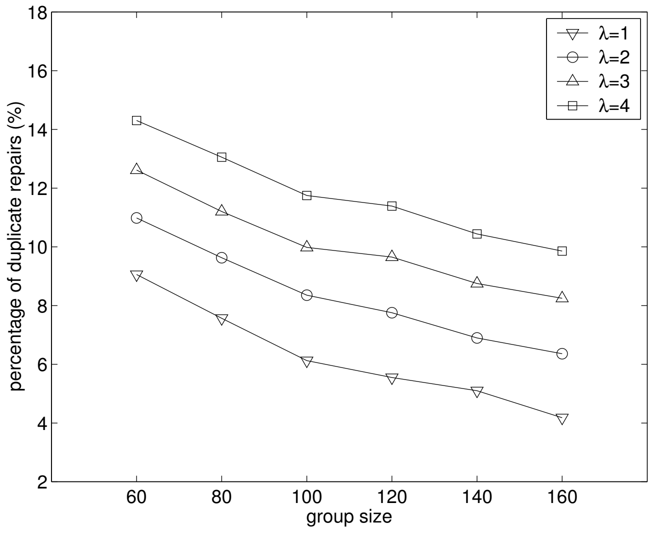
Clearly there is a trade-off between recovery latency and message duplication controlled by the parameter λ: large λ reduces recovery latency at the price of a higher number of duplicate repairs. On the other hand, small λ reduces the number of duplicate repairs but leads to longer recovery latency. This trade-off is demonstrated in Figure 7.7 for a group of 160 members when λ is increased from 1 to 4 with an increment of 0.5 at each step. The figure shows that, when $\lambda = 4$, the recovery latency of RRMP is comparable to that of TRMP, while its percentage of duplicate repairs is within 10%. We believe that this is a low overhead for enhanced robustness.

7.5 Local Recovery

As discussed in Chapter 4, one problem with a tree-based protocol is that its performance depends on the positions of repair servers. So far in the simulation tests the repair servers have been placed at the optimal positions: they are connected to

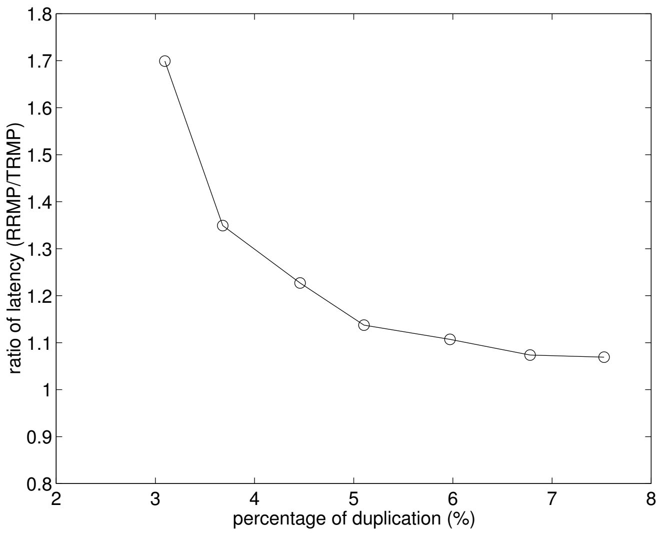


Test I: Tree topology, random loss

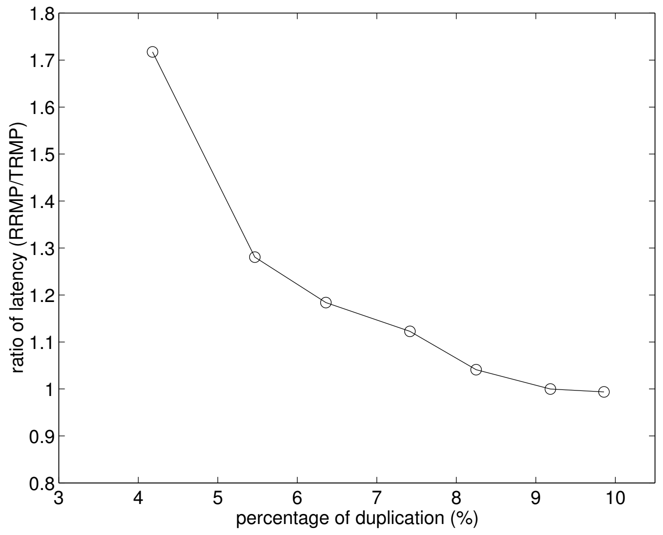


Test II: Transit-stub topology, congestion loss

Figure 7.6: Repair duplication



Test I: Tree topology, random loss



Test II: Transit-stub topology, congestion loss

Figure 7.7: Trade-off between recovery latency and repair duplication for different values of λ.

the root of the multicast subtree in their regions. However, in practice it may be difficult to predict the topological structure of the underlying multicast tree at the transport layer. If a repair server is placed at an inappropriate position, it may lead to poor locality in error recovery. We study the behavior of TRMP in such situations by moving a repair server to the left branch of the multicast tree in its region rather than connecting it to the root. This is illustrated in Figure 7.8.

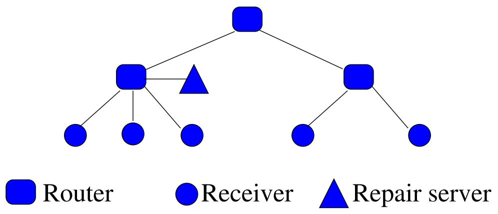
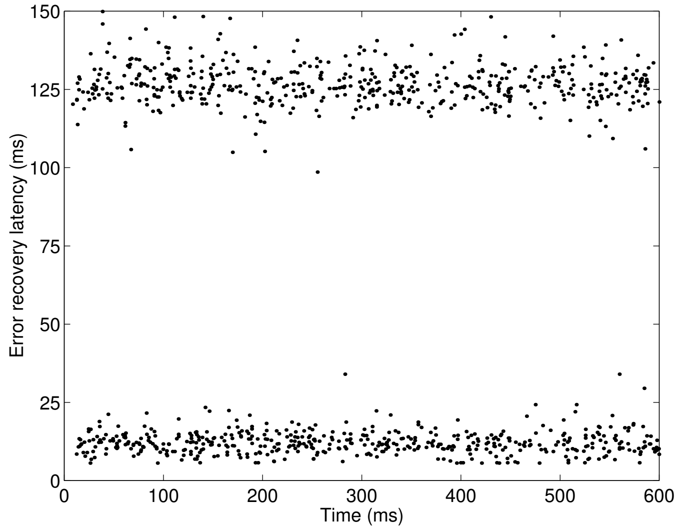


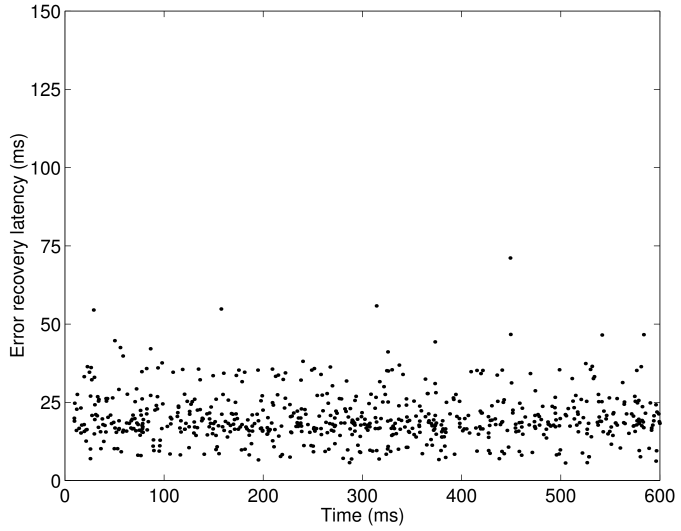
Figure 7.8: Suboptimal position of a repair server: the server is placed at the left branch of the multicast tree in its region rather than being connected to the root.

We introduce background traffic by establishing FTP applications between randomly selected nodes within each region. However, no background traffic is introduced on links connecting two regions. Consequently, every message loss in this simulation is local: it affects only a small fraction of receivers in a region. Ideally, this message loss should be recovered within the local region. We measure the error recovery latency at a receiver downstream from the repair server and show the results in Figure 7.9. Each dot in the figure represents one message loss. The x-axis shows the time when the loss occurred and the y-axis shows the corresponding error recovery latency.

The top figure indicates that the recovery latency for the TRMP protocol falls into two ranges: when a message loss occurs on links between the receiver and its repair server, it can be recovered fairly quickly. In contrast, when a loss occurs on



(a) TRMP



(b) RRMP

Figure 7.9: TRMP has poor locality in error recovery when the repair servers are placed at suboptimal positions. In contrast, a local message loss in RRMP is always recovered locally.

links between the repair server and the root of the multicast subtree in its region, it takes substantially longer to recover because the repair server needs to solicit a retransmission from its upstream repair server in the hierarchy. The bottom figure shows that for the RRMP protocol a message loss can always be recovered within a local region.

7.6 Conclusion

This chapter has compared the performance of the RRMP protocol with that of a tree-based protocol using simulation. Both protocols divide receivers in a multicast group into local regions and organize these regions into an error recovery hierarchy. However, simulation results indicate that the two protocols behave in very different ways:

- In a tree-based protocol, all receivers in a local region rely on a single repair server for error recovery. Consequently, the load on the repair server increases with the size of the region. In contrast, RRMP achieves better load balancing by diffusing the responsibility of error recovery among all receivers in the group. A receiver suffering a high loss rate can put a heavy burden on its repair server for a tree-based protocol, but has only a limited impact on its neighbors for the RRMP protocol.

- There is a tunable trade-off between error recovery latency and repair duplication for the RRMP protocol. With appropriate configuration parameters, the protocol can achieve comparable performance with a tree-based protocol while providing enhanced robustness against process failures.

- The performance of a tree-based protocol is sensitive to the positions of its repair servers and may suffer from poor locality in error recovery in certain situations. In contrast, RRMP does not use any repair server. Its concurrent execution of the local recovery phase and the remote recovery phase makes it highly likely that a local message loss will be recovered from a local member.

In summary, our work has demonstrated that randomization is a powerful technique to achieve high robustness and efficiency in reliable multicast communications.

Chapter 8

Experimental Results

This chapter evaluates the performance of the RRMP protocol on the UNIX platform. Since a major design goal of the protocol is to achieve efficient error recovery on a large multicast group, ideally we want to test the software over a wide-area network. However, doing so requires significant administrative overhead in obtaining guest accounts and installing the software at several institutions. Consequently, we decide to emulate a wide area network on a group of UNIX workstations inside the Computer Science Department at Cornell University. The topological structure of the network is described in a configuration file. The emulator intercepts the communication between the software and the network. It emulates wide area links by injecting artificial delay and message loss into the communication. This allows us to study the behavior of the protocol on a virtual network created using the local computing facility.

The error recovery algorithm in the RRMP protocol combines our previous work on randomized error recovery in the Bimodal Multicast protocol [BHO$^+$99] and hierarchical error recovery similar to that employed by tree-based protocols. The previous chapter has compared the performance of RRMP with that of tree-based protocols. In this chapter, we compare the performance of RRMP with that of Bimodal Multicast. The two protocols differ in the following ways:

- Bimodal Multicast is designed for "many-to-many" multicast applications and does not use any hierarchical structure in its error recovery. Consequently, the protocol will have a scalability problem in genuinely large networks. In contrast, RRMP focuses on "one-to-many" applications and proposes an algorithm for establishing an error recovery hierarchy based on geographic locations of different receivers.

- In Bimodal Multicast, a member exchanges its message history with other members only at fixed intervals. Hence a member missing a message has to wait until it receives history information from another member naming the message before it can send a retransmission request. In contrast, the features of the RRMP protocol include the concurrent execution of the local recovery phase and the remote recovery phase as soon as a message loss is detected and the dynamic measurements of round trip time to related members.

- In Bimodal Multicast, a member buffers received messages for a fixed amount of time after their initial reception and then garbage collects the message. In contrast, the RRMP protocol optimizes buffer requirements through a two-phase buffering algorithm that explicitly addresses the variances in delivery latency for large multicast groups. A member uses the retransmission requests it received as feedback to allocate buffer space adaptively to messages most needed in the system. The load of buffering is spread evenly among all members in the group.

8.1 Test Description

We conduct three sets of experiments to study the behaviors of the RRMP protocol and the Bimodal Multicast protocol. The first set of experiments compares the amount of error control traffic sent on wide area links between the two protocols, the second set of experiments compares their error recovery latency, and the third set of experiments compares their buffer requirements. The parameter λ for RRMP is set to 4. The length of a gossip round for Bimodal Multicast is 100ms. There are two topologies used in the experiments:

- *a single LAN*: In this topology, all members in the multicast group are in a local area network. They communicate directly over the underlying physical network without intervention of the emulator.
- *a two-LAN cluster*: In this topology, members in the group are divided evenly across two local regions. Messages sent within a local region experience the normal delay of the underlying physical network. Messages sent between the two regions have an additional delay of 30ms to emulate wide area links.

In both topologies, the group has one sender which multicasts $1K$ byte messages at a rate of 100 messages per second. Messages are delivered to the application in FIFO order. We introduce independent, random message loss with probability 1% at each member (except the sender). This is in addition to any message loss experienced in the underlying physical network.

8.2 Inter-region traffic

First we compare the amount of traffic sent on wide area links between the two protocols. The experiment consists of a group of 30 members spread evenly in two local regions. We measure the number of error control messages (gossips, requests, and repairs) sent between the two regions. The results are shown in Figure 8.1. The x-axis shows the times when the measurements were taken and the y-axis shows the number of error control messages per second. As can be seen from the figure, Bimodal Multicast introduces a high volume of inter-region traffic. This is because its error control strategy ignores topological information of the underlying network: a member selects its gossip destination uniformly at random from all other members in the group. In contrast, each region in the RRMP protocol only generates a constant number of error control messages outside the region. Hence it substantially reduces the traffic load on wide area links.

In the second experiment, we increase the group size from 8 members to 30 members with an increment of 2 at each step. For each group size the set of members are distributed evenly between the two regions. We investigate the impact of the group size increase on inter-region traffic. Figure 8.2 shows that the amount of traffic climbs rapidly with the Bimodal Multicast protocol, while it remains relatively constant for the RRMP protocol.

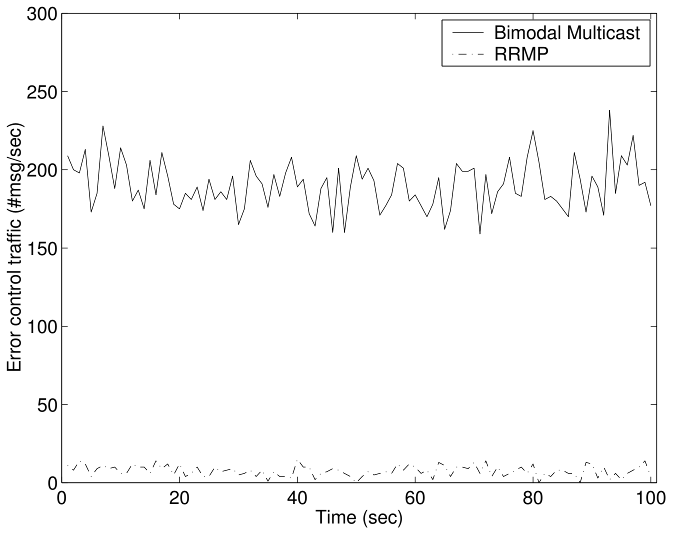


Figure 8.1: Comparison of error control traffic on wide area links between Bimodal Multicast and RRMP in a group of 30 members spread evenly in two local regions.

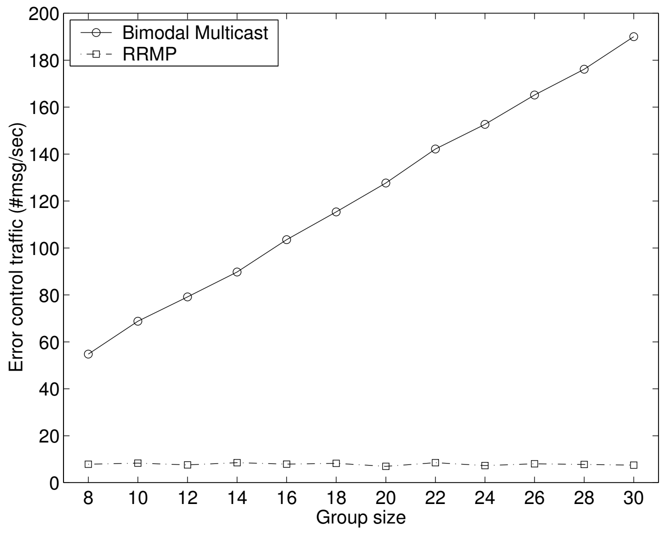
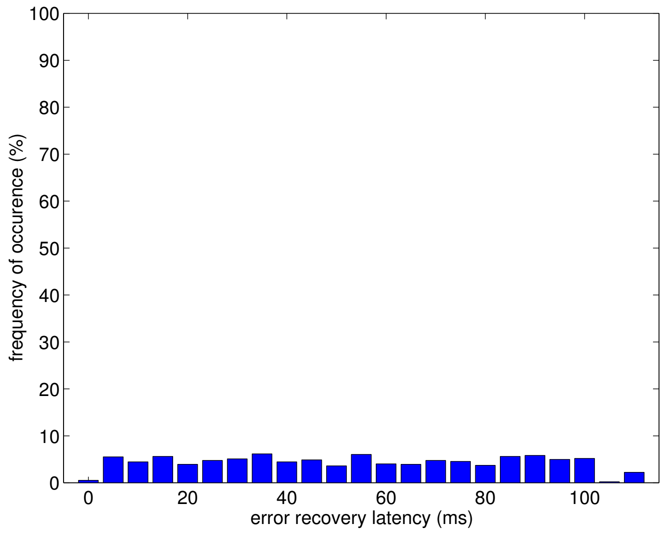


Figure 8.2: Comparison of error control traffic on wide area links between Bimodal Multicast and RRMP when the group size increases from 8 members to 30 members.

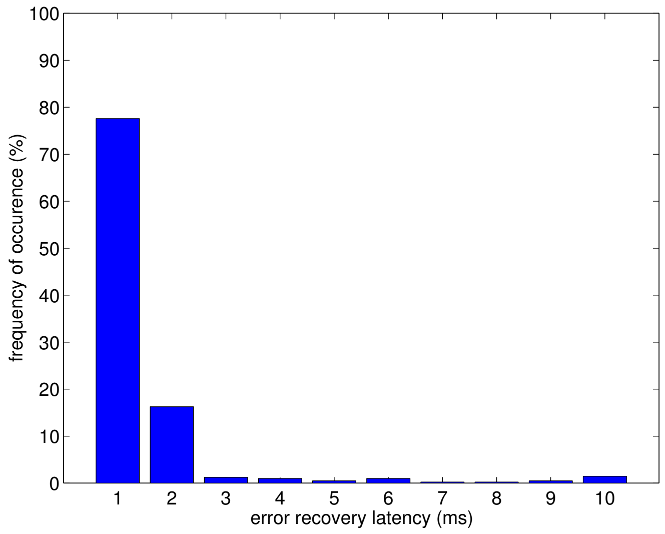
8.3 Error recovery latency

We measure the interval between when a receiver observes a gap in its sequence number space and when the corresponding missing message is recovered. We compare the error recovery latency in Bimodal Multicast with that in RRMP. We start with a group of two members in a local area network, one of which is the sender. The sender is sending $1K$ byte messages at a rate of 100 messages per second. We introduce random message loss with probability 1% at the receiver. Figure 8.3 shows the distribution of error recovery latency for the two protocols. The top figure indicates that the latency for the Bimodal Multicast protocol spreads over a wide range. This is because each member exchanges message history with other members only at fixed intervals (100ms in the current implementation). If a message is lost immediately after the sender has sent its message history to the receiver, the receiver needs to wait until the next gossip round before it can solicit a retransmission. If message losses occur uniformly over time, on average the receiver needs to wait for 50ms before sending a request. Since the latency between two machines in a local area network is typically much smaller than 50ms, the gossip interval becomes the dominant factor. In contrast, a receiver in the RRMP protocol starts sending requests as soon as it detects a message loss. The bottom figure indicates that most message losses are recovered very quickly.

An obvious solution to improve the performance of the Bimodal Multicast protocol is to reduce the gossip round interval so that members in the group exchange message history more frequently. Although this allows a receiver to solicit a retransmission faster, it increases the amount of network bandwidth consumed by gossip



(a) Bimodal Multicast



(b) RRMP

Figure 8.3: Comparison of error recovery latency between Bimodal Multicast and RRMP in a local area network. The group consists of one sender and one receiver. The scales on the x-axis of the two figures are different.

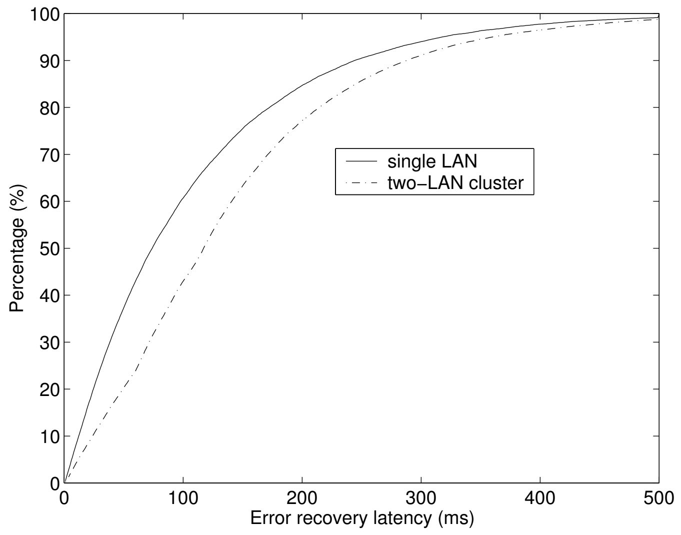
messages. This is especially problematic in a wide area network as demonstrated by results shown in the previous section.

In the second experiment, we increase the group size to 30 members and investigate its impact on error recovery latency. Figure 8.4 shows the results on two topologies: a single LAN and a two-LAN cluster. The x-axis is the error recovery latency in milliseconds and the y-axis is the percentage of message losses which are recovered within the corresponding amount of time.

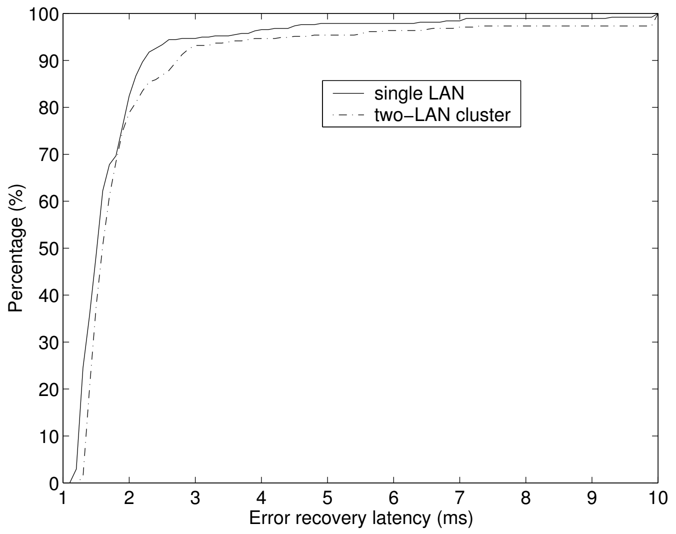
The figure indicates that error recovery latency in the Bimodal Multicast protocol (the top figure) is significantly higher than that in the RRMP protocol (the bottom figure). In particular, a substantial fraction of message losses in the Bimodal Multicast protocol take longer than one gossip round to recover even when all members are in a local-area network. This is due to the randomized nature of its gossip algorithm. Recall that a member in the Bimodal Multicast protocol sends its history of received messages to a randomly selected member in the group. On average, a member can expect to receive a gossip message from some other member in each round. However, it is possible that a member does not receive any gossip message due to randomization. Let n be the size of the multicast group and p the probability that a member receives no gossip message in a particular round. We have:

$$p = (1 - \frac{1}{n-1})^{n-1}$$

Since a member makes an independent choice to select its gossip destination in each round, the probability that a member receives no gossip message in k consecutive rounds is p^k. Table 8.1 shows this probability for different values of n and k. For example, the table indicates that when the group has 32 members the probability



(a) Bimodal Multicast



(b) RRMP

Figure 8.4: Comparison of error recovery latency between Bimodal Multicast and RRMP in a group of 30 members. The scales on the x-axis of the two figures are different.

for a member to receive no gossip message in two consecutive rounds is 13.1%. Consequently, messages lost during this period may take a long time to recover. On the other hand, it is also possible for a member to receive multiple gossip messages in one round, in which case the error recovery latency may be reduced.

Table 8.1: Probability that a receiver receives no gossip message in k consecutive rounds for different group sizes.

	$n = 4$	$n = 8$	$n = 16$	$n = 32$	$n = 64$	$n = 128$
$k = 1$	29.6%	34.0%	35.5%	36.2%	36.5%	36.6%
$k = 2$	8.8%	11.6%	12.6%	13.1%	13.3%	13.4%
$k = 3$	2.6%	3.9%	4.5%	4.7%	4.9%	4.9%
$k = 4$	0.8%	1.3%	1.6%	1.7%	1.8%	1.8%

When the 30 members are divided evenly into two local regions, error recovery latency for the Bimodal Multicast protocol has increased noticeably. This is because the protocol does not utilize any topological information of the multicast group. Consequently, a member missing a message may solicit a retransmission from a remote member even if the lost message can be recovered from its neighbors. In contrast, the RRMP protocol organizes members in the group into an error recovery hierarchy based on their geographical locations. A member missing a message concurrently executes both the local error recovery phase and the remote error recovery phase. This increases the likelihood that a local message loss will be repaired by a local member. The bottom figure shows that the resulting error recovery latency for both topologies remains very low.

8.4 Buffer Requirements

Finally we compare the amount of buffer requirements in the Bimodal Multicast protocol with that in the RRMP protocol. The experiment was conducted in a group of 30 members in a local area network. The sender sends $1K$ byte messages at a rate of 100 messages per second. We introduce random message loss with probability 1% at each receiver and show the results in Figure 8.5. The x-axis indicates the times when the measurements were taken and the y-axis indicates the number of messages a member keeps in its buffer.

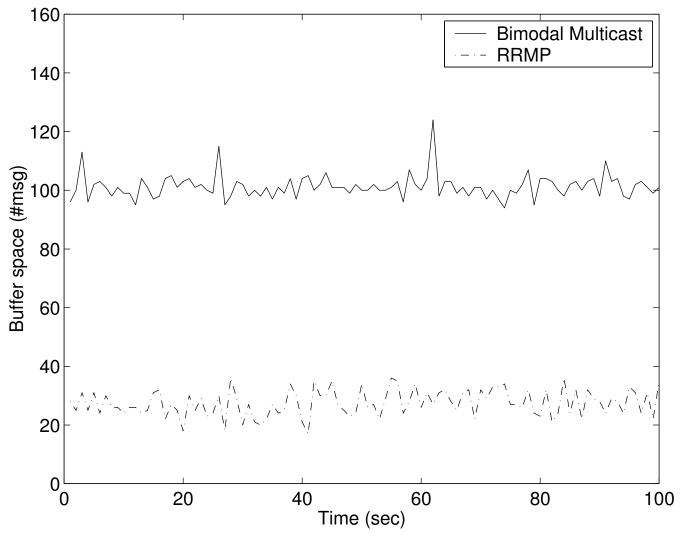


Figure 8.5: Comparison of buffer requirements between Bimodal Multicast and RRMP in a group of 30 members in a local area network.

Recall that the Bimodal Multicast protocol uses a simple buffering policy where a member buffers received messages for a fixed amount of time (10 rounds in the current implementation). The figure shows that the number of messages in a member's buffer is around 100. In contrast, the RRMP protocol divides its buffer space into two parts:

a short-term buffer and a long-term buffer. When a member first receives a message, it keeps the message in its short-term buffer until no request for this message has been received for a certain period of time (50ms in the current implementation). Then the member makes a random choice to become a long-term bufferer with probability C/n. In this experiment, we set $C = 6$ and $n = 30$. Hence on average 20% of the members in a region serve as long-term bufferers. A long-term bufferer keeps the message for 1 second. The figure shows that the resulting buffer requirements are substantially smaller than that for the Bimodal Multicast protocol.

The amount of buffer space in the Bimodal Multicast protocol can be reduced if a member buffers received messages for a shorter period of time. In order to be comparable to the RRMP protocol, a member should buffer a message for approximately 250ms. However, Figure 8.4 shows that a noticeable fraction of message losses in the Bimodal Multicast protocol may take longer than 250ms to recover. Consequently, the application will experience a higher loss rate.

One concern with the two-phase buffering scheme in the RRMP protocol is its potential negative impact on error recovery latency: after a message has become idle (i.e. no request for this message has been received for 50ms), only a subset of members in a local region will continue to buffer the message. If a member discards a message and then later receives a retransmission request for that message, it cannot answer the request itself and needs to search for a bufferer of the message. This is usually not a problem when all members are in a local area network, or when message losses occur randomly and independently, because Figure 8.4 demonstrates that error recovery latency in this case is much smaller than 50ms. The situation is quite different in a wide area network where all members in a region may miss the same message. To study the behavior of the RRMP protocol under such situations, we conduct another experiment in which the 30 members are divided evenly into two local regions. Messages sent between the two regions have a delay of 30ms and a random loss probability of 5%. Messages sent within a local region have no loss. Because all members in the downstream region will miss the same message, the lost message can only be repaired through the remote recovery phase. Due to the long latency between the two regions (the round trip time is larger than 50ms), a member in the sender's region may receive a remote request from a downstream member, asking for a message that it has already discarded. In this case it needs to search for a bufferer of the message.

As a target for comparison, we implemented a *single-phase* buffering scheme in which all members continue to buffer an idle message for 1 second (i.e. every member is a long-term bufferer). We compare the error recovery latency between the two schemes and show the results in Figure 8.6. The x-axis is the error recovery latency in milliseconds and the y-axis is the percentage of message losses that are recovered within the corresponding amount of time. The figure indicates that the two-phase buffering scheme incurs only a small performance penalty in error recovery latency while providing a substantial reduction in buffer requirements.

8.5 Conclusion

This chapter has compared the performance of RRMP with that of Bimodal Multicast on the UNIX platform. The RRMP protocol is inspired by the Bimodal Multicast protocol and inherits its idea of randomized error recovery. The major

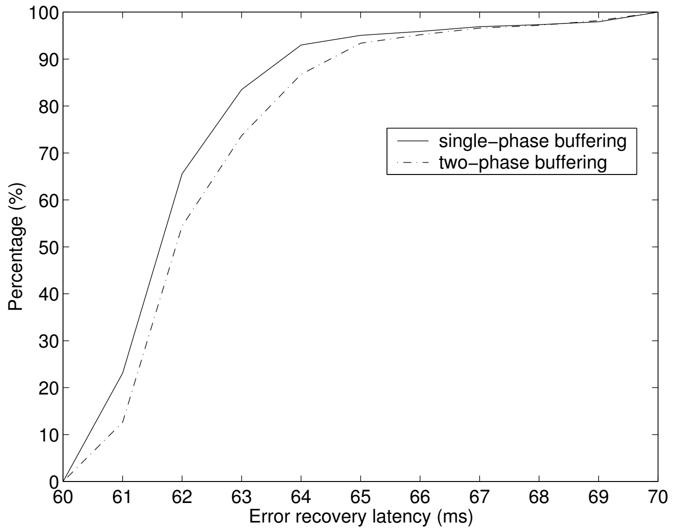


Figure 8.6: Comparison of error recovery latency with two buffering schemes.

improvements include the use of an error recovery hierarchy to achieve better scalability on a wide-area network, the concurrent execution of local recovery and remote recovery to reduce recovery latency, and an adaptive two-phase buffering scheme to optimize buffer requirements. Experimental results have demonstrated that these improvements can have a significant impact on the performance of the protocol.

Chapter 9

Conclusion and Future Work

This dissertation has studied the problem of providing reliable multicast service in large groups. We started by discussing several challenges in the design of efficient error recovery algorithms:

- How to avoid message implosion?
- How to confine the impact of a message loss to the region where the loss has occurred?
- How to manage buffer space efficiently?

We then examined previous work in the reliable multicast literature and found that existing protocols only partially addressed these challenges. For example, the SRM protocol avoids message implosion through a randomized back-off algorithm at the expense of increased error recovery latency. In addition, it suffers from a *crying baby* problem due to its lack of local recovery. A tree-based protocol provides local recovery by organizing receivers into local regions and selecting a repair server for each region. The protocol still suffers from a regional implosion problem because a repair server bears the entire responsibility of error recovery for a local region. Moreover, the deployment of repair servers requires topological information of the underlying multicast tree which is not available at the transport layer. Router-assisted reliable multicast protocols achieve high performance and scalability by utilizing new functionalities at network routers. Whether these extra functionalities can become widely deployed remains to be seen.

After an overview of previous work, we described the Bimodal Multicast protocol developed in part by the author. The protocol achieves superior scalability over virtual synchrony protocols through random gossiping technique. In this protocol, members periodically exchange history of received messages with randomly selected members and solicit any message they have missed. Experimental results demonstrate that the protocol has strong throughput properties and is highly robust in the presence of network failures.

We then proceeded into the main focus of the dissertation: the Randomized Reliable Multicast Protocol (RRMP). RRMP organizes receivers into a hierarchy of local regions without imposing any specific structure inside a region. The hierarchy formation algorithm is based on periodic exchange of global session messages and local session messages among all receivers in the group. It allows a receiver to obtain an approximation of receivers in its local region as well as receivers in its parent region. The error recovery algorithm in RRMP consists of the concurrent execution of the local recovery phase and the remote recovery phase: a receiver missing a message sends its request to randomly selected receivers in its local region and, with a small probability, to some randomly selected receiver in a remote region. The protocol eliminates message implosion because the responsibility of error recovery is disseminated among all receivers in the group. In addition, the concurrent execution of the two error recovery phases increases the likelihood that a local message loss will be repaired by a local member. The reliability of the protocol depends on statistical properties of its randomized algorithm which were formally analyzed and can be tuned according to application requirements.

RRMP optimizes buffer management through an innovative two-phase buffering scheme: feedback-based short-term buffering and randomized long-term buffering. When a message is first introduced into the system, every member that receives the message buffers it for a short period of time in order to satisfy retransmission requests from its neighbors. Later when the message has been received by almost all members in a region, only a small subset of members in this region continue to buffer the message. The algorithm effectively reduces buffer requirements by adaptively allocating buffer space to messages most needed in the system and by spreading the load of buffering among all members in the group.

Looking ahead, there are several directions for future research. Error recovery in the RRMP protocol is retransmission-based. Recently, Forward Error Correction (FEC) has been proposed in several reliable multicast protocols as an efficient technique for providing error recovery of uncorrelated loss in large multicast groups [McA90, Riz97, NBT98]. In these protocols, the sender takes k packets from the application and generates n encoded packets in such a way that any subset of k encoded packets are sufficient to reconstruct the original packets. The advantage of this approach is that a receiver can recover from up to $n - k$ packet losses without the need to ask for retransmissions. Hence it creates an illusion of a network substantially more reliable than the underlying physical network. Moreover, the sender can simultaneously repair distinct message losses at different receivers by multicasting a single encoded packet to the group. Hence this approach has the potential of significantly reducing bandwidth consumption due to error control messages. It would be interesting to investigate how FEC techniques could be incorporated into the RRMP protocol to further improve its scalability.

A major problem in constructing an efficient error recovery hierarchy involves estimating the geographical locations of different receivers. A solution proposed in [RM99a, RM99b] is to infer the topological structure of the underlying multicast tree based on message loss correlation across the receiver set. In this algorithm, each receiver records the sequence of messages it has missed. This is called the *lossprint* of the receiver. Since a message loss along an upper link of a multicast distribution tree will cause all downstream receivers to miss the same message, two receivers that share a great portion of their multicast paths from the sender are likely to have a strong correlation in their message losses. Receivers in the group periodically exchange lossprints to compute their loss correlation. The algorithm groups receivers with similar loss patterns into a loss neighborhood so that the scope of error recovery can be localized within this neighborhood without disturbing the rest of the group[1]. A nice feature of this approach is that it relies solely on information available at the transport level without requiring additional support from network routers. Consequently, it can be integrated into protocols like RRMP as a component for constructing the error recovery hierarchy. Moreover, information about loss

[1]The paper does not propose a specific protocol to perform error recovery. Rather the algorithm is designed as a component that can be used by reliable multicast protocols to construct an error recovery hierarchy.

correlation at different receivers can be used to guide the error recovery process. For example, a receiver missing a message may use such information to decide where to send its retransmission requests. It would be interesting to see how such a protocol would perform in practice.

BIBLIOGRAPHY

[ADKM92] Yair Amir, Danny Dolev, Shlomo Kramer, and Dalia Malki. Transis: A communication sub-system for high availability. In *International Symposium on Fault-Tolerant Computing*, July 1992.

[AMMS$^+$95] Y. Amir, L. E. Moser, P. M. Melliar-Smith, D. A. Agarwal, and P. Ciarfella. The Totem single-ring ordering and membership protocol. In *ACM Transactions on Computer Systems*, November 1995.

[AP99] Mark Allman and Vern Paxson. On estimating end-to-end network path properties. In *Proceedings of ACM SIGCOMM*, 1999.

[Bai75] Norman Bailey. *The Mathematical Theory of Infectious Diseases and its Applications.* Hafner Press, 1975.

[BHO$^+$99] Kenneth P. Birman, Mark Hayden, Oznur Ozkasap, Zhen Xiao, Mihai Budiu, and Yaron Minsky. Bimodal multicast. In *ACM Transactions on Computer Systems*, May 1999.

[Bir93] Kenneth P. Birman. The process group approach to reliable distributed computing. In *Communications of ACM*, December 1993.

[Bir97] Kenneth P. Birman. *Building Secure and Reliable Network Applications.* Manning Publishing Company and Prentice Hall, 1997.

[BJ87a] Kenneth P. Birman and Thomas A. Joseph. Exploiting virtual synchrony in distributed systems. In *ACM Symposium on Operating Systems Principles*, 1987.

[BJ87b] Kenneth P. Birman and Thomas A. Joseph. Reliable communication in the presence of failures. In *ACM Transactions on Computer Systems*, February 1987.

[BvR94] Kenneth P. Birman and Robbert van Renesse. Reliable distributed computing with the ISIS toolkit. In *IEEE Computer Society Press*, 1994.

[CDZ97] Kenneth Calvert, Matthew Doar, and Ellen Zegura. Modeling Internet topology. In *IEEE Communications Magazine*, June 1997.

[CM99] Adam M. Costello and Steven McCanne. Search Party: Using randomcast for reliable multicast with local recovery. In *Proceedings of IEEE INFOCOM*, 1999.

[CT90] David D. Clark and David L. Tennenhouse. Architectural considerations for a new generation of protocols. In *Proceedings of ACM SIGCOMM*, 1990.

[DC90] Stephen E. Deering and David R. Cheriton. Multicast routing in datagram internetworks and extended LANs. In *ACM Transactions on Computer Systems*, May 1990.

[DGH$^+$87] Alan Demers, Dan Greene, Carl Hauser, Wes Irish, John Larson, Scott Shenker, Howard Sturgis, Dan Swinehart, and Doug Terry. Epidemic algorithms for replicated database maintenance. In *ACM Symposium on Principles of Distributed Computing*, 1987.

[Dur94] Richard Durrett. *The Essentials of Probability.* Duxbury Press, 1994.

[FJM$^+$95] Sally Floyd, Van Jacobson, Steven McCanne, Ching-Gung Liu, and Lixia Zhang. A reliable multicast framework for light-weight sessions and application level framing. In *Proceedings of ACM SIGCOMM*, 1995.

[For95] XTP Forum. Xpress transfer protocol specification. In *XTP Rev 4.0*, March 1995.

[GR00] Katherine Guo and Injong Rhee. Message stability detection for reliable multicast. In *Proceedings of IEEE INFOCOM*, 2000.

[GT92] Richard Golding and Kim Taylor. Group membership in the epidemic style. Technical report, University of California at Santa Cruz, 1992.

[Han97] Mark Handley. An examination of MBone performance. In *ISI Research Report ISI/RR-97-450*, April 1997.

[Hay98] Mark Hayden. *The Ensemble System.* Ph.D. dissertation, Cornell University, January 1998.

[Hof97] Markus Hofman. Enabling group communication in global networks. In *Proceedings of Global Networking*, 1997.

[HSC95] Hugh Holbrook, Sandeep Singhal, and David Cheriton. Log-based receiver-reliable multicast for distributed interactive simulation. In *Proceedings of ACM SIGCOMM*, 1995.

[Jac88] Van Jacobson. Congestion avoidance and control. In *Proceedings of ACM SIGCOMM*, 1988.

[Ler00] Xavier Leroy. The Object Caml system release 3.00, April 2000. http://pauillac.inria.fr/ocaml/.

[LESZ98] Ching-Gung Liu, Deborah Estrin, Scott Shenker, and Lixia Zhang. Local error recovery in SRM: Comparison of two approaches. In *IEEE/ACM Transactions on Networking*, December 1998.

[LLSG92] Rivka Ladin, Barbara Liskov, Liuba Shrira, and Sanjay Ghemawat. Providing availability using lazy replication. In *ACM Transactions on Computer Systems*, November 1992.

[LOM94] Kurt Lidl, Josh Osborne, and Joseph Malcome. Drinking from the firehose: Multicast USENET news. In *Proceedings of USENIX Winter Conference*, January 1994.

[LPGLA98] Brian Neil Levine, Sanjoy Paul, and J.J. Garcia-Luna-Aceves. Organizing multicast receivers deterministically by packet-loss correlation. In *Proceedings of ACM Multimedia*, 1998.

[McA90] Anthony J. McAuley. Reliable broadband communication using a burst erasure correcting code. In *Proceedings of ACM SIGCOMM*, 1990.

[Mil91] David L. Mills. Internet time synchronization: The network time protocol. In *IEEE Transactions on Communications*, October 1991.

[NBT98] Jorg Nonnenmacher, Ernst W. Biersack, and Don Towsley. Parity-based loss recovery for reliable multicast transmission. In *IEEE/ACM Transactions on Networking*, May 1998.

[OD81] Derek C. Oppen and Yogen K. Dalal. The Clearinghouse: A decentralized agent for locating named objects in a distributed environment. Technical report, Xerox, 1981.

[OvRBX99] Oznur Ozkasap, Robbert van Renesse, Kenneth P. Birman, and Zhen Xiao. Efficient buffering in reliable multicast protocols. In *International Workshop on Networked Group Communication*, November 1999.

[Pax97a] Vern Paxson. Automated packet trace analysis of TCP implementations. In *Proceedings of ACM SIGCOMM*, 1997.

[Pax97b] Vern Paxson. End-to-end Internet packet dynamics. In *Proceedings of ACM SIGCOMM*, 1997.

[Pit87] Boris Pittel. On spreading a rumor. In *SIAM Journal of Applied Mathematics*, February 1987.

[PPV98] Christos Papadopulos, Guru Parulkar, and George Varghese. An error control scheme for large-scale multicast applications. In *Proceedings of IEEE INFOCOM*, 1998.

[PSLB97] Sanjoy Paul, Krishan Sabnani, John Lin, and Supratik Bhattacharyya. Reliable multicast transport protocol (RMTP). In *IEEE Journal on Selected Areas in Communication, special issue on Network Support for Multipoint Communication*, 1997.

[Rei94] Michael K. Reiter. Secure agreement protocols: Reliable and atomic group multicast in Rampart. In *ACM Conference on Computer and Communications Security*, November 1994.

[Riz97] Luigi Rizzo. Effective erasure codes for reliable computer communication protocols. In *ACM Computer Communication Review*, April 1997.

[RM99a] Sylvia Ratnasamy and Steven McCanne. Inference of multicast routing trees and bottleneck bandwidths using end-to-end measurements. In *Proceedings of IEEE INFOCOM*, 1999.

[RM99b] Sylvia Ratnasamy and Steven McCanne. Scaling end-to-end multicast transports with a topologically-sensitive group formation protocol. In *International Conference on Network Protocols*, 1999.

[SEFZ98] Puneet Sharma, Deborah Estrin, Sally Floyd, and Lixia Zhang. Scalable session messages in SRM using self-configuration. Technical report, University of Southern California, 1998.

[UCB] UCB/LBNL/VINT. network simulator ns (version 2). http://www-mash.cs.berkeley.edu/ns/.

[vRBM96] Robbert van Renesse, Kenneth P. Birman, and Silvano Maffeis. Horus: A flexible group communication system. In *Communications of ACM*, April 1996.

[vRMH98] Robbert van Renesse, Yaron Minsky, and Mark Hayden. A gossip-style failure detection service. In *Proceedings of Middleware*, 1998.

[XB01a] Zhen Xiao and Kenneth P. Birman. Providing efficient, robust error recovery through randomization. In *International Workshop on Applied Reliable Group Communication*, April 2001.

[XB01b] Zhen Xiao and Kenneth P. Birman. A randomized error recovery algorithm for reliable multicast. In *Proceedings of IEEE INFOCOM*, April 2001.

[YGS95] Rajendra Yavatkar, James Griffioen, and Madhu Sudan. A reliable dissemination protocol for interactive collaborative applications. In *Proceedings of ACM Multimedia*, 1995.

[YKT96] Maya Yajnik, Jim Kurose, and Don Towsley. Packet loss correlation in the MBone multicast network: Experimental measurements and markov chain models. In *Proceedings of IEEE INFOCOM*, 1996.