Optimizing Buffer Management for Reliable Multicast

Zhen Xiao, Kenneth P. Birman, Robbert van Renesse*

Department of Computer Science

Cornell University

Email: {xiao, ken, rvr}@cs.cornell.edu

109 16 pages

Abstract

Reliable multicast delivery requires that a multicast message be received by all members in a group. Hence certain or all members need to buffer messages for possible retransmissions. 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.

We propose an innovative two-phase buffering algorithm, which explicitly addresses variations in delivery latency seen in large multicast groups. 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. Simulation results demonstrate that the algorithm has good performance.

1 Introduction

Multicast is an efficient way for disseminating data to a large group. Many emerging multicast applications require reliability guarantees not provided by the IP multicast protocol [5]. Providing reliable multicast service on a large scale requires an efficient error recovery algorithm. It has been shown that putting the responsibility of error recovery entirely on the sender can lead to a message implosion problem [7, 12]. Consequently, several reliable multicast protocols adopt a distributed error recovery approach which allows certain or all members to retransmit packets lost by other members. For example, in the SRM protocol [7], the Bimodal Multicast protocol [3], and the

*This work was supported in part by DARPA/RADC under grant number F30602-99-1-0532.

Randomized Reliable Multicast Protocol [14], retransmissions are performed by all members in the group. In tree-based protocols like RMTP [12], LBRRM [9], and TMTP [15], 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. This 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) [4]. This requires that the application be designed according to the ALF principle and is capable of reconstructing packets. In addition, buffer management at the application level is still 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 [8]. This requires members in the group to periodically exchange message history information about the set of messages they have received. In addition, a failure detection algorithm is needed to provide current group membership information.

Previously we proposed a message buffering algorithm for reliable multicast protocols which reduces the amount of buffer requirement by only buffering messages on a small set of members [11].

More specifically, we assume that each member has an approximation of the entire membership in the form of network addresses[1]. 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[2]. 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. The probability of this happening and the associated penalty in latency both increase with the size of the group. Hence the protocol will have a scalabilty problem in genuinely large networks. Desired is an algorithm which 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 here was motivated by this observation.

In this paper, we report our work on optimizing buffer requirements for a randomized reliable multicast protocol called RRMP. The error recovery algorithm in RRMP combines our previous work on randomized error recovery in the Bimodal Multicast protocol [3] and hierarchical error recovery similar to that employed by tree-based protocols. Its buffer management extends our previous work on buffer optimization by proposing 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. Unlike stability detection protocols, the algorithm does not require periodic exchange of messages and has low traffic overhead.

The rest of the paper is organized as follows. In Section 2 we briefly describe the error recovery algorithm in RRMP protocol because it is closely related to the work reported here. A complete description can be found in [14]. Readers already familiar with the protocol can proceed directly to Section 3, where we describe the details of our buffer management algorithm. Simulation results are presented in Section 4. Limitations of the algorithm are presented in Section 5. Section 6 concludes.

[1] The approximation need not be accurate, but it should be of good enough quality so that the probability of the group being logically *partitioned* into disconnected subgroups is negligible.

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

2 A Randomized Error Recovery Algorithm

The RRMP protocol is designed for multicast applications with only one sender. Traditionally tree-based protocols [12, 9, 15] have been proposed for such applications. In [14] we demonstrated certain performance problems in tree-based protocols because the responsibility of error recovery in each local region is concentrated on a single repair server. RRMP improves the robustness and efficiency of tree-based protocols by diffusing the responsibility of error recovery among all members in the group. It was built on our previous work of the Bimodal Multicast protocol [3] and the Gossip-style Failure Detection protocol [13]. However, the Bimodal Multicast protocol uses a simple buffering policy in which each member buffers messages for a fixed amount of time.

2.1 System Model and Assumptions

We assume that receivers are grouped into local regions and different regions are organized into a hierarchy according to their distance from the sender. We call this the "error recovery hierarchy". The sender joins the multicast group before it starts sending messages, and consequently is also a receiver in the group. Figure 1 shows an example of a hierarchy where the whole group is divided into three local regions. We define the *parent* region of a receiver as its least upstream region in the hierarchy. For example, in Figure 1, 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 has group membership knowledge about other receivers in its region as well as receivers in its parent region.

A receiver detects a message loss by observing a gap in the sequence number space. In addition, session messages are used to help a receiver detect the loss of the last message in a burst.

2.2 Randomized Error Recovery

Unlike tree-based protocols, RRMP does not use any repair server. The responsibility of error recovery is randomly distributed among all members 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 get a retransmission from a neighbor 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

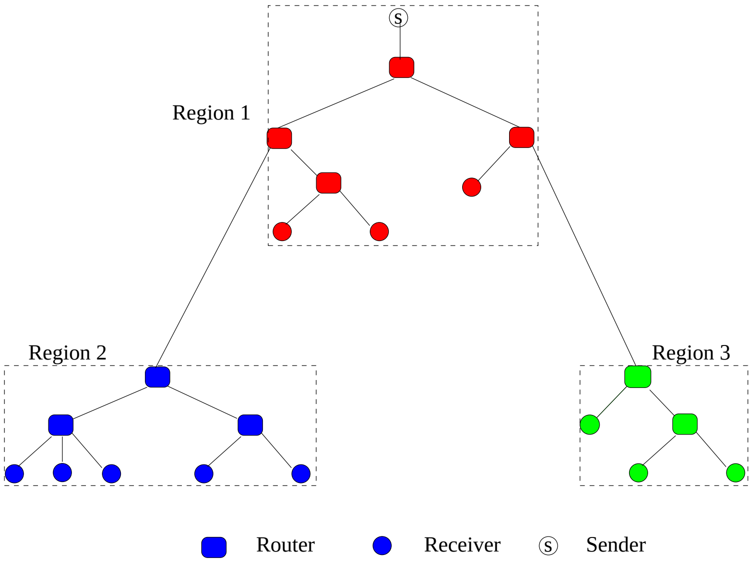

Figure 1: *Local regions in a hierarchical structure*

repaired through local recovery, while a regional loss is repaired by a combination of local recovery and remote recovery. The rest of the subsection 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. 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 able to recover the loss eventually. This has been shown in previous work on epidemic theory [1, 10]. In particular, a receiver in the sender's region is able to recover any message loss 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, sends a request to r. This probability is chosen so that the expected number of remote requests sent by all receivers in the region is a constant λ. For example, if $\lambda = 1$, then on average one remote request is sent when the entire region missed a message. 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.

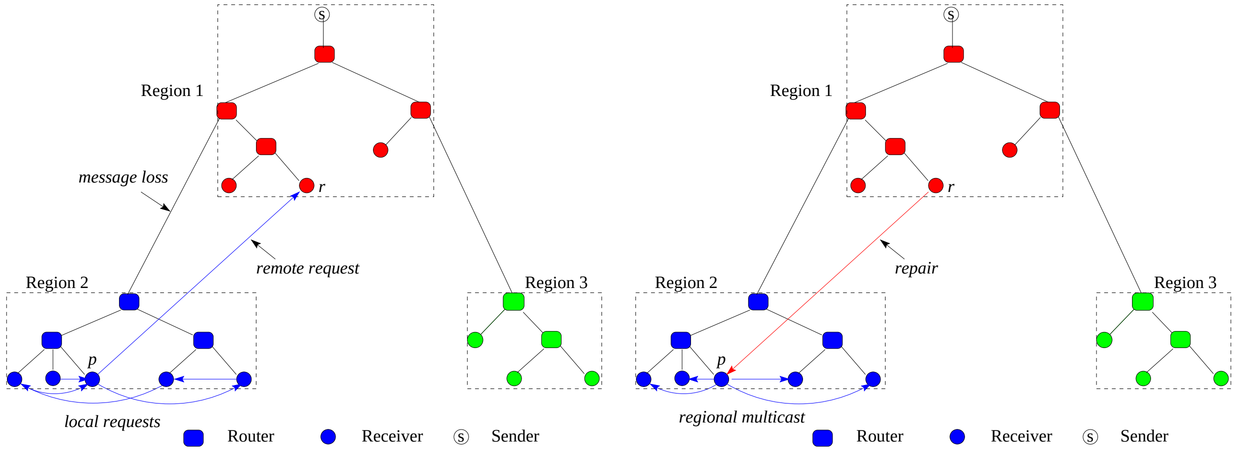

Figure 2: *Error recovery in RRMP*

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.

If multiple members in a region simultaneously receive repairs from upstream members for the same message, all of them will multicast the repair in this region. In [14] we suggested a randomized back-off scheme to suppress duplicate regional multcast at the expense of longer recovery latency.

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 2 illustrates RRMP's error recovery algorithm when all receivers in region 2 missed a message. On the left, local requests are sent to randomly selected neighbors, and one of them, p, sends a request to a remote member r. On the right, member r forwards a copy of the message to p, which then multicasts the message in its local region.

3 Optimizing Buffer Management

As described in the previous section, 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 tradeoff 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.

3.1 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 about 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 [8]. 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 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 sufficiently long 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.

3.2 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 which 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 [6]. 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 [3]. The probability that k members buffer an idle message is $e^{-C}\frac{C^k}{k!}$. Figure 3 shows how the distribution changes with different values of C. The choice of C reflects a tradeoff 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 4. When $C = 6$, for example, the probability is only 0.25%.

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.

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.

3.3 Search for Bufferers

When a member p receives a remote request from a downstream member r for a message, there are 3 possibilities:

[3]In [14] we applied a similar technique to analyze the number of remote requests sent when an entire region missed a message.

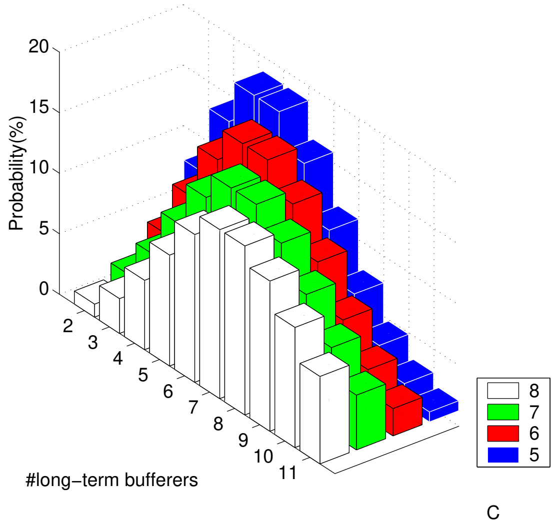

Figure 3: *The probability that k members buffer an idle message for different values of C.*

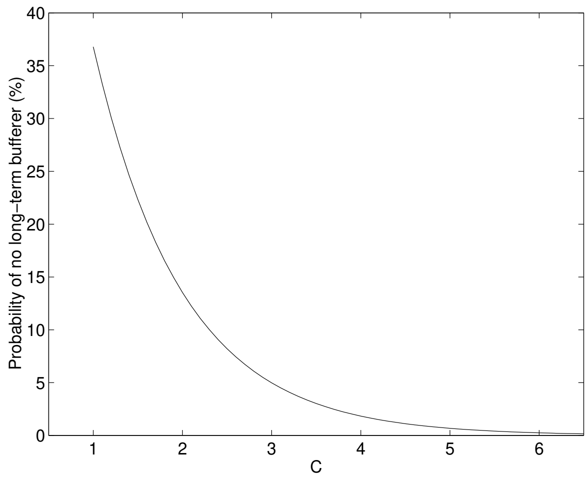

Figure 4: *The probability that no member buffers an idle message decreases exponentially with C.*

- 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 Section 2. 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 from another member, it suppress its own multicast. The problem is how to choose the 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 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 [4]. 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 5 illustrates the search process in a region with 4 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 5 ms. 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 10 ms 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.

The above discussion is simplified in assuming that p is the only member receiving a remote request. As described in Section 2, 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.

[4] If q has never received the message, it will send retransmission requests as described in Section 2.

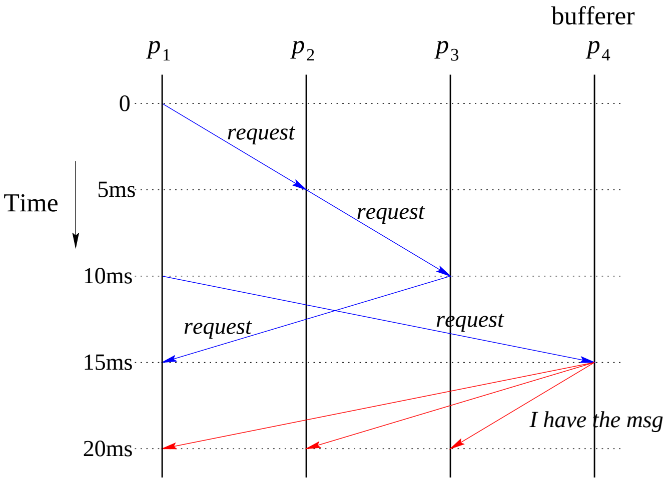

Figure 5: *Search for bufferers in a local region*

3.4 Comparison

In RRMP protocol, the set of long-term bufferers are chosen randomly from all receivers in a region. Previously we proposed a deterministic algorithm [11] which chooses a subset of receivers in a group to serve as bufferers using a hash function as described in Section 1. It is interesting to compare these two approaches.

We believe the choice between them 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 incurred during the search process but has higher computation overhead.

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 with a deterministic algorithm.

4 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 10 ms. The idle threshold T is set to 40 ms (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 (note that the y-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.

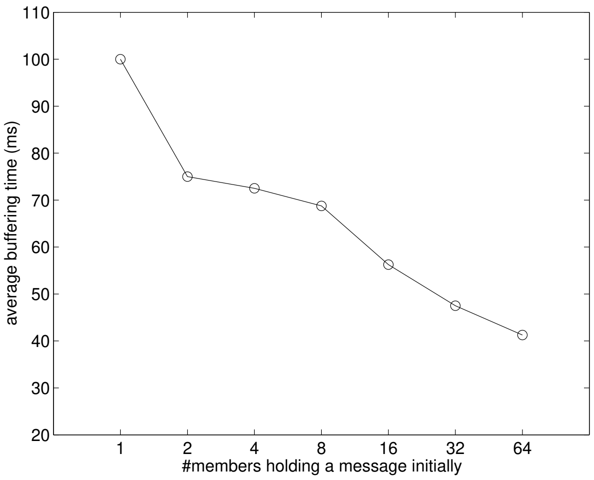

Figure 6: *Effectiveness of feedback-based buffering. The y-axis is in logarithmic scale.*

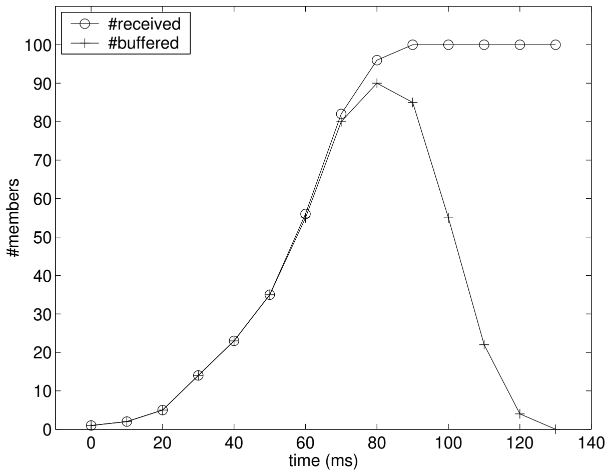

Figure 7: *Comparison between the number of members which have received a message and the number of members which buffer the message as error recovery proceeds.*

In Figure 7 we take a closer look at one of the data points in Figure 6 where 1 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 decline 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.

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 8 shows that the search time decreases as the number of bufferers increases [5]. With 10 bufferers, for example, the average search time is 20 ms (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 9 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. 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.

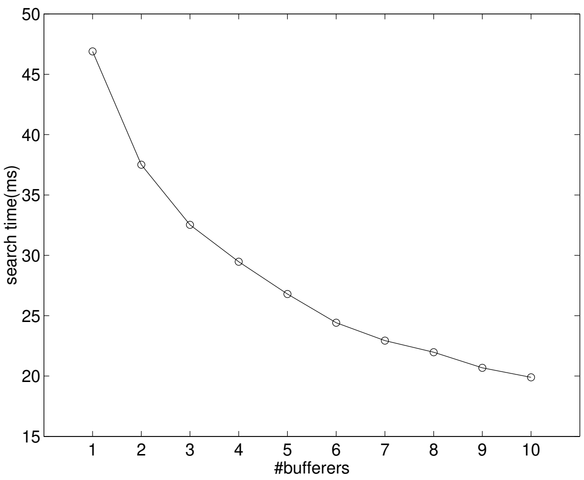

Figure 8: *Search time as the number of bufferers increases.*

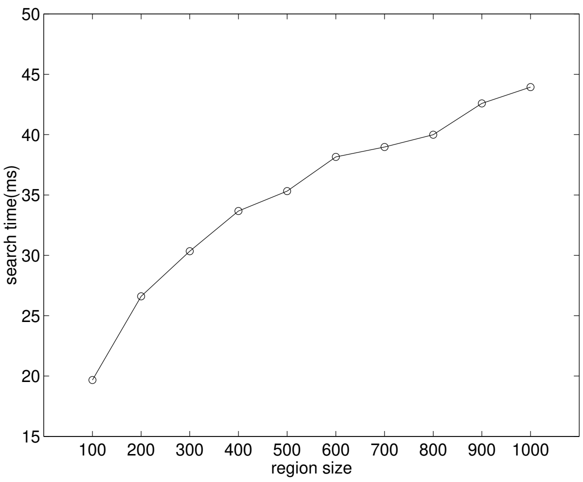

Figure 9: *Search time as the size of the region increases.*

5 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

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

made arbitrarily small with carefully chosen parameters for the protocol, but still must be accounted for when designing an application.

Applications which require a stronger guarantee should use a protocol that provides end-to-end reliability, such as virtual synchrony [2]. The probabilistic guarantees offered by RRMP have the benefit of superior scalability and intrinisic robustness in networks subject to message loss and process failures, but are not appropriate when absolute guarantees of reliability are needed.

6 Conclusion

Buffer management is essential to a reliable multicast protocol. This paper presents an innovative two-phase buffering algorithm in a randomized reliable multicast protocol called RRMP which provides efficient buffering for large multicast groups. It extends our previous work on buffer optimization by explicitly addressing issues that arise in a wide area network. 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. In the future we plan to investigate how the techniques described in this paper can be incorporated into other reliable multicast protocols.

References

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

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

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

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

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

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

[7] 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.

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

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

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

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

[12] 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.

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

[14] Zhen Xiao and Kenneth P. Birman. A randomized error recovery algorithm for reliable multicast. Technical report, Department of Computer Science, Cornell University, May 2000. http://www.cs.cornell.edu/home/xiao/rrmp.ps.

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