# A Randomized Error Recovery Algorithm for Reliable Multicast

Zhen Xiao, Kenneth P. Birman

*Abstract*— **An efficient error recovery algorithm is essential for reliable multicast in large groups. Tree-based protocols (RMTP, TMTP, LBRRM) group receivers into local regions and select a repair server for performing error recovery in each region. Hence a single server bears the entire responsibility of error recovery for a region. In addition, the deployment of repair servers requires topological information of the underlying multicast tree, which is generally not available at the transport layer.**

**This paper presents RRMP, a randomized reliable multicast protocol which improves the robustness of tree-based protocols by diffusing the responsibility of error recovery among all members in a group. The protocol works well within the existing IP multicast framework and does not require additional support from routers. Both analysis and simulation results show that the performance penalty due to randomization is low and can be tuned according to application requirements.**

## I. Introduction

Multicast is an efficient way for distributing data from one sender to multiple receivers. The existing IP multicast protocol [1] is unreliable. However, many applications 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. 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) [2]. 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.

Another challenge in the design of error control algorithm 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 [3], [4]. If retransmission requests and replies are multicasted 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 [5].

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 [6]), the Log-Based Receiver-Reliable Multicast Protocol (LBRRM [7]), and the Tree-based Multicast Transport Protocol(TMTP [8]). 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.

However, a tree-based protocol may still suffer from a regional implosion problem because the responsibility of error recovery in each region is concentrated on a single repair server. Consequently, a receiver experiencing a high loss rate can put a heavy burden on its repair server and affect all other receivers in its region. Moreover, if a repair server fails, error recovery in its region cannot proceed until a new server is selected.

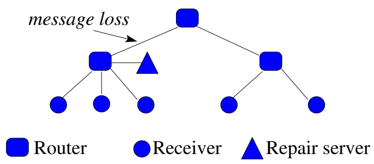



Fig. 1. *Performance penalty in a tree-based protocol when the position of a repair server is suboptimal.*

Another 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, topological information of the underlying multicast tree is generally not available at the transport layer. Improper positioning of repair servers can lead to poor locality in error recovery. Figure 1 shows a portion of a multicast tree in a region where the repair server is placed at the left branch of the tree rather than at the root. A message loss as indicated caused 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 outside this local region (not shown in the figure), leading to long recovery latency[1].

Previously we proposed a "Bimodal Multicast" protocol for "many-to-many" multicast applications [9] [10]. In this protocol, a member $p$ periodically sends its history of received messages to a randomly selected member $q$ in the group. Upon receiving the message history, $q$ asks $p$ any message that $q$ missed but $p$ has received so that $p$ and $q$ can converge to identical histories. Message retransmissions are performed in the order of most recent first. Hence the protocol emphasizes achieving a common suffix of message histories rather than a common prefix. If a message loss cannot be recovered after a certain amount of time, the protocol gives up on the message and reports the loss to the application. It is shown that the 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.

In this paper, 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, but focuses on how to use randomization to improve the robustness and efficiency of tree-based protocols. A detailed comparison between RRMP and Bimodal Multicast is deferred to Section VI so that sufficient background can be established.

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 analysed and tuned according to application requirements. Simulation results demonstrate that the protocol achieves fast error recovery with low overhead, compared to tree-based protocols.

The rest of the paper is organized as follows. Section II describes the details of our error recovery algorithm. Section III analyses its performance. Section IV describes an algorithm for constructing the error recovery hierarchy. Simulation results are presented in Section V. Section VI describes related work. Section VII concludes.

## II. A Randomized Error Recovery Algorithm

### A. 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 2 shows an example of a hierarchy where the whole group is divided into three local regions. Section IV describes an algorithm for constructing such a hierarchy. We define the *parent* region of a receiver as its least upstream region in the hierarchy[2]. For example, in Figure 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 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 or by exchanging session messages with other members (described in Section IV).

In this paper, we focus on the error recovery algorithm in RRMP. Other aspects of the protocol (e.g. buffer management) is reported in a separate paper [11]. For simplicity, we assume that every receiver buffers received mes-

[1] 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.

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

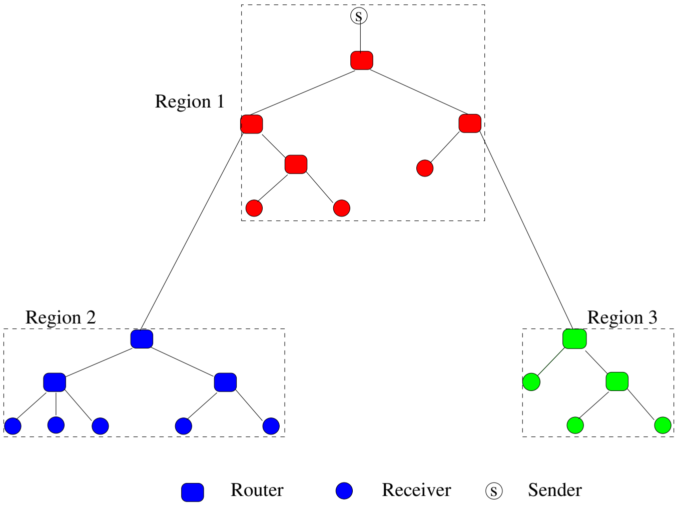



Fig. 2. *Local regions in a hierarchical structure*

sages for a sufficiently long period of time. In practice, a receiver with insufficient storage capacity may choose not to perform error recovery for other receivers.

## *B. Details of the Algorithm*

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 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 $q$. Round trip time measurements are described in the next subsection. 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. 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 $\lambda$. 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 $\lambda$.

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 3 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.

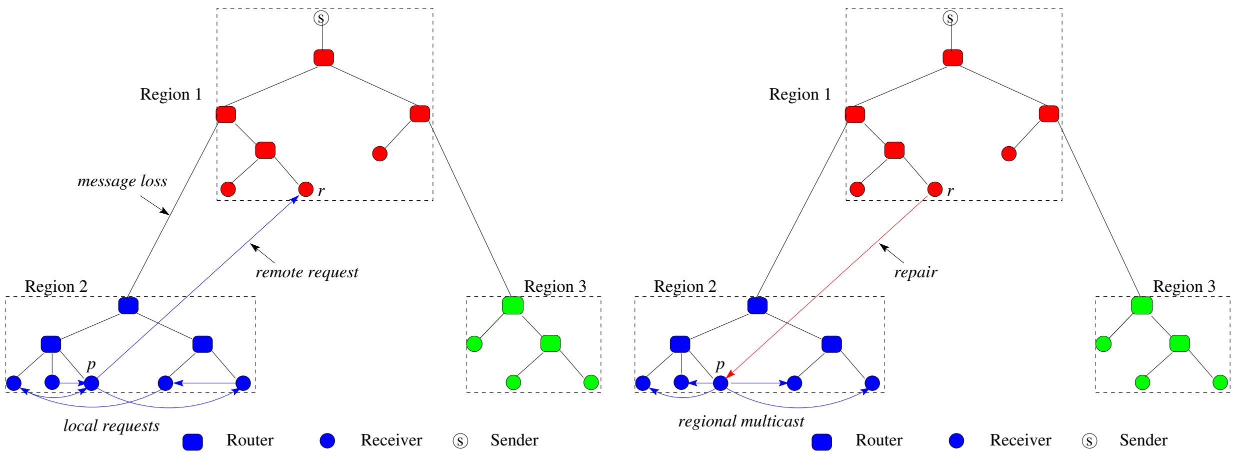



Fig. 3. *Error recovery in RRMP*

### C. 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. 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 [12].

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 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 SRM protocol. More specifically, when a member $r$ sends a repair to a remote member $p$, it includes two timestamps, $t$ and $\Delta$, where $t$ is the timestamp copied from $p$'s request, and $\Delta$ 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. 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.

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 tradeoff between bandwidth consumption and accuracy of RTT measurements.

### D. Optimization

In this subsection, we describe two optimizations of the basic error recovery algorithm. The first optimization aims to reduce 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 3. Assume that $p$'s RTT to $r$ is 100ms and to a local member is 1ms. In this case $p$ will 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 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 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.

The second optimization aims to reduce repair traffic. When all members in a region missed a message, on average $\lambda$ 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 duplication in this case is independent of $n$ but increases with $\lambda$. 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 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. Both optimizations are incorporated in the simulation later in this paper.

## III. Performance Analysis

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

### A. 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 no failure of a single member can have a significant impact on other members in the group.

### B. Recovery Latency

The 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 [13], [14]. Figure 4 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 formula used to compute this figure is omitted for lack of space. Recovery latency can be reduced if a member sends multiple requests at a time, although this may increase the number of duplicate repairs.

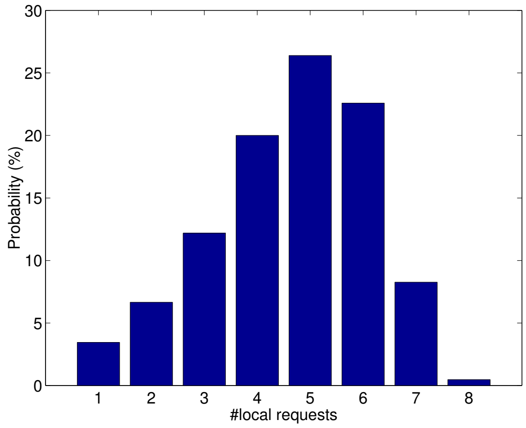



Fig. 4. *The probability for a member to receive a repair in a particular request for a region of* 30 *members, with one member holding the message initially.*

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 $\lambda$ [15][3]. The probability that $k$ requests are sent is $e^{-\lambda}\frac{\lambda^k}{k!}$. Figure 5 shows how the distribution changes with different values of $\lambda$. The choice of $\lambda$ reflects a tradeoff between recovery latency and repair duplication. As shown in Figure 5, when $\lambda$ is small, there is a substantial risk that no remote request is sent due to randomization, leading to increased recovery latency. Increasing $\lambda$ reduces this risk and improves robustness against loss of request and repair messages. On the other hand, large $\lambda$ increases the number of duplicate repairs as explained in the following subsection.

[3]A similar observation is made in the Search Party protocol [16], although the details are very different.

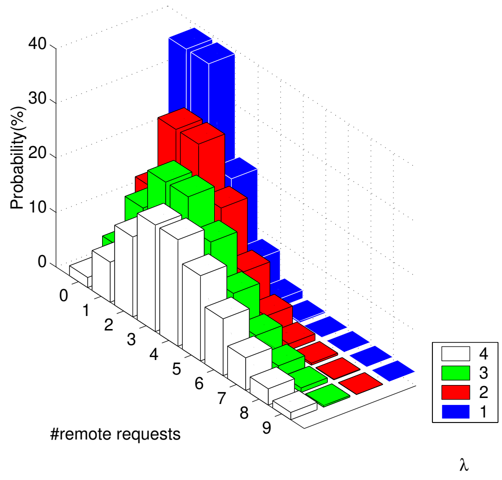



Fig. 5. *The probability that $k$ remote requests are sent for different values of $\lambda$.*

The concurrent execution of local recovery phase and remote recovery phase increases the likelihood that a local message loss will be repaired by a local member. This is especially important when inter-region latency is much higher than intra-region latency. For example, in Figure 1, the receivers at the left branch of the tree are likely to get retransmissions from the two receivers at the right branch, thus avoiding inter-region latency.

### C. Repair Duplication

Duplicate repairs can be received for various reasons. For example, 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. In addition, the concurrent execution of local recovery and remote recovery may trigger multiple repairs for the same message: upon detection of a loss, a member sends a remote request with probability $\lambda/n$. 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 $\lambda$. Hence $\lambda$ controls a trade-off between recovery latency and repair duplication: large $\lambda$ reduces recovery latency at the price of higher number of duplicate repairs. On the other hand, small $\lambda$ reduces the number of duplicate repairs but leads to longer recovery latency. Applications with different delay/bandwidth requirements can tune $\lambda$ to suit their own needs.

### D. Locality of Recovery

Locality of recovery can be measured by comparing the number of members receiving a repair to 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.

## IV. Formation of Error Recovery Hierarchy

### A. Overview

This section describes an algorithm for constructing the error recovery hierarchy in RRMP. The algorithm groups receivers into local regions based on administrative domains and organize 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. There are two kinds of session messages: *local* session messages and *global* session messages[4]. Local session messages are multicast restricted within a local region and global session messages are multicast to 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. The global session interval $T_{gs}$ and the local session interval $T_{ls}$ are configuration parameters of the system.

### B. 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. Members within a region periodically exchange local session messages to learn about their neighbors and to estimate the size

[4]The terminology *local* and *global* session messages are borrowed from the scalable session message protocol in SRM [17], but the algorithm is different.

of the region. A local session message from a receiver $p$ contains the largest sequence number $p$ has received from the sender (for loss detection purpose). Soft state timers are used to detect receivers which have left the region.

### C. 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, a member $r$ multicasts a global session message only with probability $\lambda'/n$ during an interval of $T_{gs}$, where $\lambda'$ is a system configuration parameter and is not necessarily the same as the $\lambda$ used for error recovery. On average $\lambda'$ global session messages are sent per region. The global session message contains the largest sequence number $r$ has received from the sender, its hop counts from the sender, and the initial TTL value. A member obtains its hop counts from the sender through the TTL field of data messages it received from the sender.

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. $r$ is upstream from $p$ if $r$ is closer to the sender than $p$ and $p$ is closer to $r$ than to the sender. All distance information used in the comparison can either be calculated by $p$ or is included in $r$'s global session message. 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 $r$, and 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. $H$ is a system configuration parameter which controls the degree of heterogeneity in the parent region. 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.

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.

### D. Properties of the Algorithm

In our algorithm, the parent region for a receiver does not necessarily corresponds 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. 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 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 possiblity of getting a repair when some links in the network get congested.

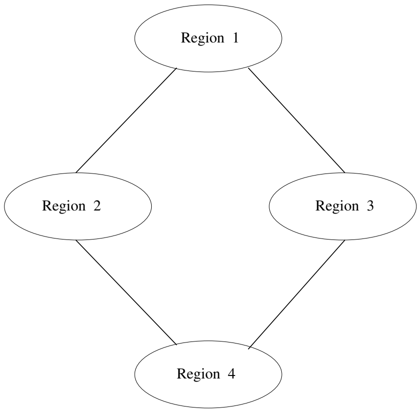



Fig. 6. *Formation of error recovery hierarchy in RRMP. The parent region of a receiver in region* $4$ *may contain a mixture of receivers from region* $2$ *and receivers from region* $3$.

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.

## V. Simulation Results

### A. Test Description

In this section, we evaluate the performance of RRMP using the ns2 simulator [18]. 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 missed a message, it gets a retransmission from its least upstream server in the hierarchy and then multicasts the retranmission 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 RRMP protocol due to randomization[5].

The topologies used in the simulation are transit-stub networks generated using the GT-ITM network generator [19]. 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. For the tree-based protocol, each stub domain also has a repair server connected to the root of that domain.

Both RRMP and TRMP use a mixture of unicast and regional multicast for repair messages. Each stub domain is assumed to be in a different administrative domain and administrative scoping is used to restrict the scope of a regional multicast. The configuration parameters for RRMP protocol is shown in Table I.

| | |
|---|---|
| $T_{maxl}$ | 1 second |
| $T_{maxr}$ | 5 second |
| $T_{gs}$ | 1 second |
| $T_{ls}$ | 1 second |
| $H$ | 4 hops |
| $\lambda'$ | 2 |

TABLE I

*Configuration parameters*

In the simulation, members exchange session messages to form the error recovery hierarchy as described in Section IV. 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 message/sec. 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.

We introduce background traffic by establishing TCP connections between randomly selected nodes. For each TCP connection, a 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. Consequently, a repair server receives any message which is received by at least one member in its region. This is the optimal case for a tree-based protocol.

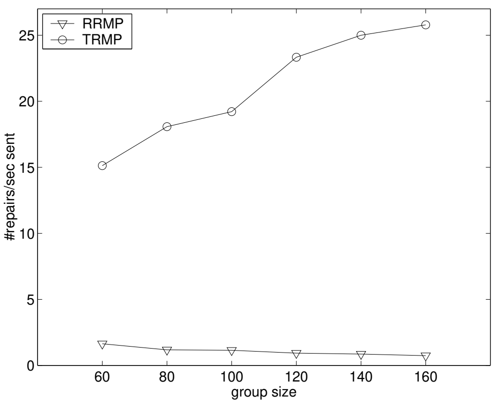
### B. Load Balance

First we compare the load of request and repair traffic between the two protocols. The results are shown in Figure 7. On the left we compare the number of repair messages sent by a repair server in TRMP protocol with the maximum number of repair messages sent by a member in RRMP protocol during the simulation. On the right we compare the number of request messages received by a repair server in TRMP protocol with that by the worst-case member in RRMP protocol. The parameter $\lambda$ for RRMP protocol is set to 4. As can be seen from the figure, the load on the repair server increases linearly with the group size for TRMP protocol. This is because the repair server bears the entire burden of error recovery for its region. In contrast, the load for RRMP protocol 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 $\lambda$.

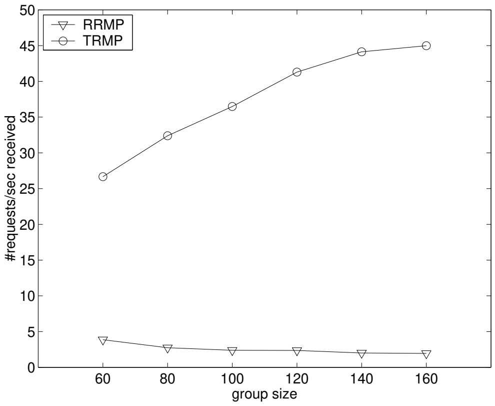
One way to reduce the load on a repair server is to split a large region into several small ones. This is effective if all members in the region have roughly the same loss rate. Otherwise a single member suffering a high loss rate can put a heavy burden on its repair server even after the split. This is shown in Figure 8 for a group of 160 members when the loss rate of one receiver is increased from $1\%$ to $28\%$ [6]. We compare the number of repair messages sent to this lossy receiver by its repair server in TRMP proto-

[5]It would be interesting to compare RRMP with some existing tree-based protocols like RMTP. Unfortunately, the source code for the RMTP protocol was not made available to us by the developers.

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




(a) repair traffic




(b) request traffic

Fig. 7. *Comparison of request and repair traffic when group size increases*

col with that by the worst-case member in RRMP protocol. (The figure for request load is similar and hence omited.) As can be seen from the figure, a lossy receiver can have a significant impact on its repair server in TRMP protocol but only a limited impact on its neighbors in RRMP protocol[7].

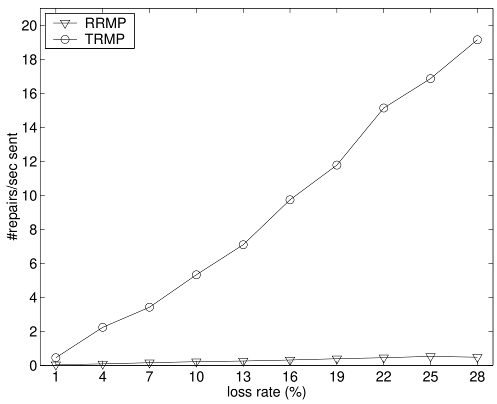



Fig. 8. *Comparison of repair traffic for a lossy receiver*

### C. Recovery Latency

Each member measures the average recovery latency over all message loss it experienced. We compute the ratio of recovery latency in RRMP to that in TRMP memberwise. Figure 9 shows the average ratio for different values of $\lambda$ when the group size increases. As can be seen from the figure, there is an observable performance penalty for RRMP protocol due to randomization when $\lambda = 1$ or 2. The latency of RRMP improves when $\lambda$ increases. When $\lambda = 4$, 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 $\lambda$, the latency of RRMP does not increase with group size, which indicates that the algorithm scales well.

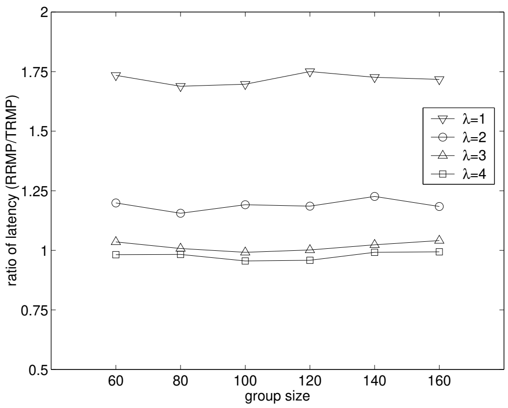



Fig. 9. *Error recovery latency*

### D. Repair Duplication

In RRMP protocol, each member calculates the percentage of repair messages it received which are duplicates. The result is averaged over all receivers in the group. Figure 10 shows that the percentage of duplication is low and decreases with group size, for any given $\lambda$. Clearly there is a tradeoff between recovery latency and message duplication. This is demonstrated in Figure 11 for a group of 160 members when $\lambda$ 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 slightly better than that of TRMP, while its percentage of duplicate repairs is about 10%. We believe that this is a low overhead for enhanced robustness.

[7] In some tree-based protocols (e.g. RMTP), a repair server multicasts a repair message if it has received several requests for that message. Again this is ineffective if a single member suffers from a high loss rate.

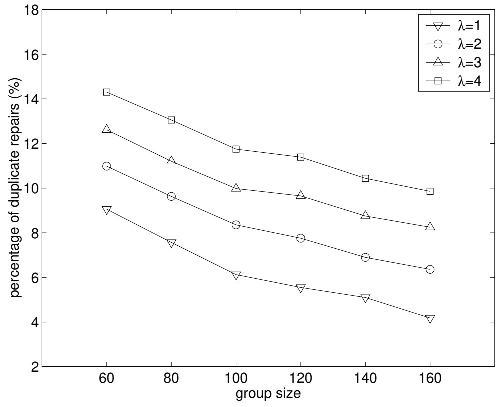



Fig. 10. *Repair duplication*

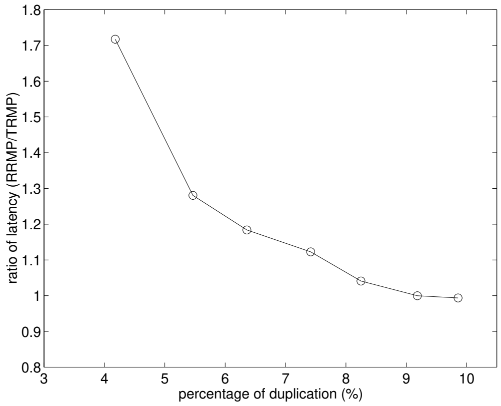



Fig. 11. *Tradeoff between recovery latency and repair duplication for different values of* $\lambda$.

### *E. Locality of Recovery*

RRMP achieved good locality of recovery in our simulation: only those members missing a message are exposed to the repair traffic. This is because the ratio of inter-region latency to intra-region latency is high in the generated topologies. Consequently, a repair is sent in regional multicast only if the entire region missed the message.

## VI. Related Work

Randomization was previously used in epidemic algorithms to disseminate updates in a distributed database environment [20][14]. More recently, van Renesse et al. proposed a failure detection service using the random gossiping technique [21].

The error recovery algorithm in RRMP protocol combines our previous work on randomized error recovery in the Bimodal Multicast protocol [9] and hierarchical error recovery similar to that employed by tree-based protocols. It is different from the Bimodal Multicast protocol in the following ways: Bimodal Multicast is designed for "many-to-many" multicast applications and 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. In addition, a member only exchanges its message history with other members 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, RRMP focuses on "one-to-many" applications and proposes an algorithm for establishing an error recovery hierarchy based geographic locations of different receivers. Its features 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.

RMTP, LBRRM, and TMTP are among the best known examples of tree-based protocols. Besides their use of repair servers, these protocols are also different from RRMP in the construction of error recovery hierarchy. Both RMTP and LBRRM are based on static hierarchy. In RMTP, for example, specific machines are chosen as repair servers and are statically organized into a tree. TMTP proposes an algorithm for dynamically organizing repair servers into a tree based on expanding ring search. In TMTP, 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.

The scalable session message protocol [17] proposes a self-configuring algorithm for establishing a hierarchical structure in a multicast group. However, the hierarchy there is used for distributing session messages in SRM protocol and not for sending retransmission requests and replies. The hierarchy is established using a stochastic algorithm based on randomized timers and a set of *appropriateness* measures. Because SRM is designed for "many-to-many" multicast applications, the hierarchy is not organized with respect to a given source.

Search Party is an error recovery protocol based on a new forwarding service called *randomcast* which forwards packets randomly inside a multicast distribution tree [16]. In this protocol, when a member $c$ 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 $c$. 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* which restricts its scope to the subtree rooted at the arrival link. A member missing a message keeps sending requests as a Poisson process until a repair arrives.

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.

## VII. Conclusions and Future Work

Error recovery is an essential part of a reliable multicast service. This paper presents 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 local recovery phase and remote recovery phase. Both analysis and simulation results show that the performance penalty due to randomization is low and can be tuned according to application requirements. At the time of this writing, we are in the middle of an Internet implementation of RRMP. We expect to release the software in the fall.

Error recovery in RRMP 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 [22], [23]. In the future, we plan to investigate how FEC can be incorporated into RRMP to further improve its scalability.

## References


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

[2] Sally Floyd, Van Jacobson, Steven McCanne, Ching-Gung Liu, and Lixia Zhang, “A reliable multicast framework for lightweight sessions and application level framing,” in *Proceedings of ACM SIGCOMM*, 1995.

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

[4] Mark Handley, “An examination of MBone performance,” in *ISI Research Report ISI/RR-97-450*, Apr. 1997.

[5] 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*, Dec. 1998.

[6] 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.

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

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

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

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

[11] Zhen Xiao, Kenneth P. Birman, and Robbert van Renesse, “Optimizing buffer management for reliable multicast,” Tech. Rep., Department of Computer Science, Cornell University, May 2000, Submitted to the Second International Workshop on Networked Group Communication.

[12] Van Jacobson, “Congestion avoidance and control,” in *Proceedings of ACM SIGCOMM*, 1988.

[13] Boris Pittel, “On spreading a rumor,” in *SIAM Journal of Applied Mathematics*, Feb. 1987.

[14] Alan Demers, Dan Greene, Carl Hauser, Wes Irish, et al., “Epidemic algorithms for replicated database maintenance,” in *Proceedings of ACM Principles of Distributed Computing*, 1987.

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

[16] Adam M. Costello and Steven McCanne, “Search Party: Using randomcast for reliable multicast with local recovery,” in *IEEE INFOCOM*, 1999.

[17] Puneet Sharma, Deborah Estrin, Sally Floyd, and Lixia Zhang, “Scalable session messages in SRM using self-configuration,” Tech. Rep., University of Southern California, 1998.

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

[19] Kenneth Calvert, Matthew Doar, and Ellen Zegura, “Modeling Internet topology,” in *IEEE Communications Magazine*, June 1997.

[20] Derek C. Oppen and Yogen K. Dalal, “The Clearinghouse: A decentralized agent for locating named objects in a distributed environment,” Tech. Rep., Xerox, 1981.

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

[22] Luigi Rizzo, “Effective erasure codes for reliable computer communication protocols,” in *ACM Computer Communication Review*, Apr. 1997.

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