Abstract

SRM is a generic framework for reliable multicast delivery. In order to maximize the collaboration among the group members in error recovery, both retransmission requests and replies are multicast to the entire group. SRM uses random timers to effectively suppress duplicate requests and replies. However, a few members with frequent losses can still cause frequent retransmissions to all the group members.

To further improve the scalability of SRM, one must localize the scope of error recovery traffic. In this paper we present two approaches to local recovery: hop-based scope control and use of local recovery groups. The first approach uses hop count to limit the distance requests and replies travel, whereas the second approach confines error recovery traffic to be within local recovery groups. The local recovery groups and hop count settings are automatically created and dynamically adjusted based on observed loss patterns. We use simulation experiments to examine the performance of both approaches.

1 Introduction

The Scalable Reliable Multicast framework (SRM) [1] is a generic framework for reliable multicast delivery. In order to maximize the collaboration among the group members in the error recovery process, both retransmission requests and replies are multicast to the entire group. SRM takes a receiver-driven approach in error recovery to avoid the message implosion problem [2], and uses random timers to effectively suppress duplicate requests and replies. Unfortunately, because every lost packet results in a reply being sent to the entire group, a single lossy link can still cause frequent global retransmissions. This behavior limits the scalability of SRM as network and group size increases. To further improve the scalability of SRM, we wish to localize the scope of error recovery traffic.

In this paper, we present two different mechanisms to localize the scope of error recovery traffic. The hop-scoped error recovery mechanism uses hop count to limit the distance that request and reply messages can travel. In contrast, the group-scoped error recovery mechanism confines the propagation of error recovery traffic...
within some local multicast groups. Simulation results of these mechanisms suggest that they both reduce error recovery traffic without introducing significant overhead.

The paper is organized as follows: Section 2 gives a brief description of SRM framework and the problems that we address in this paper. Section 3 and 4 describe the hop-scoped error recovery and group-scoped error recovery mechanisms, respectively. Section 5 presents the simulation models and analyzes the simulation results, and Section 6 reviews related works. We conclude in Section 7 with a short summary.

2 Basic Approaches of SRM

In this section, we give an overview of SRM, emphasizing the features crucial to our proposed mechanisms. We use the term *session* to mean a multicast application that uses SRM as its underlying reliable multicast service. SRM provides basic reliability support, i.e., it guarantees data delivery to all members in the multicast session. Other functionalities, such as total ordering and fate sharing, if desired, are the responsibilities of the application itself [3]. To avoid message implosion in the error recovery process, SRM is receiver-initiated [4] with each receiver being responsible for detecting data losses and requesting retransmissions. SRM also adopts the approach of “multicasting everything” to maximize the collaboration among members in the process of error recovery. Requests and replies are multicast to all members in the session. Multicasting a request allows the nearest member with the requested data to send a reply first; it also suppresses other members from sending out duplicate requests for the same data. Similarly, multicasting a reply gets the reply to all members who suffer the loss without requiring the replier to knows their exact locations, as well as suppresses duplicate replies.

The SRM mechanisms can be decomposed into two parts: group state synchronization and receiver-initiated error recovery. Members periodically exchange session messages to report current group state (e.g., the highest received sequence number from each source) and to determine the propagation delays between each pair of members. Members use group state information to detect data losses. It is critical for the receiver-initiated error recovery approach because members do not otherwise know what has been sent to the session group. Members use propagation delays to schedule their request or reply timers as described below.

When a packet gets lost, each member detecting the loss waits a random time period before sending the retransmission request. The random timer is scheduled between the time interval of \( A \cdot T \sim (A + B) \cdot T \). \( T \) is the propagation delay between the requester and the data source, and \( A \) and \( B \) are constants. The request timer interval is a function of the distance to the source because we want a member near the source to request first. When the timer expires, the scheduled request is multicast to the session group. If a duplicate request is received or the current scheduled request is sent, the requester exponentially backs off its request timer. If

\[1\] Fate sharing is when a multicast session terminates if a single member, or a specific subset of members in the session fail, depending on the semantics of the application.
a reply is received, the scheduled request is canceled. A member with the requested data \(^2\) responds to the request by scheduling a reply. The reply is scheduled in the time interval \(a \cdot t \sim (a + b) \cdot t\), where \(t\) is the propagation delay between the replier and the requester, and \(a\) and \(b\) are constants. When the timer expires, the scheduled reply is multicast to the session. The scheduled reply is canceled if a duplicate reply is received. Therefore, the member immediately behind the lossy link \(^3\) is most likely to send its request; and the member immediately above the lossy link is most likely to send its reply.

The randomization of request and reply timers in SRM gives members an opportunity to suppress one another and thus avoid the request and reply message implosion problem. However, a member with persistent losses can still trigger enough request and reply activity to overwhelm other members in the session. For example, consider the case where member \(p\) in Figure 1 loses a packet at link \(L_1\). Its request and the reply from member \(q\) will reach all members in the session, which causes duplicate data reception for all other members. Moreover, multiple lossy links may make the problem worse because multiple lossy links reduce duplicate suppression. For example, in Figure 1, if the reply from member \(q\) is lost at link \(L_2\), the scheduled reply at member \(r\) will not be suppressed and a duplicate reply is multicast to the session. In a session of size \(n\), if \(k\) members lose a packet, the error recovery traffic reaches all \(n\) members by using global error recovery. As a result, \(\frac{(n-k)}{n}\%\) of the error recovery traffic is wasted. The premise of this paper is that, an error recovery mechanism must not only avoid message implosion, but should also isolate error recovery traffic to the required scope.

We have experimented with two different mechanisms to localize error recovery traffic: hop-scoped error recovery and group-scoped error recovery. Note that local recovery is a performance optimization, thus the mechanisms do not have to achieve the optimal or precise degree of locality; the more local the recovery, the less recovery traffic overhead there is. However, independently from how well our local recovery mechanisms work, all data losses are eventually recovered. In particular, in both mechanisms that we propose here, a member may occasionally sends its requests

---

\(^2\)SRM assumes all session members, not only the data source, save all the application data. If some members do not save the data requested, they simply do not participate in the error recovery process.

\(^3\)We refer to a lossy link as the place where a data packet is dropped, it can be either a link or a router along the delivery path.
and replies to an inappropriate scope. Such occasional abnormal behavior is soon corrected and have little impact on the overall performance. We describe these two approaches in detail in the following sections.

3 Hop-Scoped Error Recovery

The simplest solution to control the scope of requests and replies is to limit the number of hops they travel. We wish to use the minimum hop counts possible in requests and replies. In order to recover from a loss, a request must reach as least one member who has the requested data. A member behind a lossy link could perhaps learn form its past experience to identify the lossy link and send its subsequent requests with a hop count that reaches beyond the lossy link. However, if there are multiple lossy links along the path, a conservative approach would be to set the hop limit large enough so that requests can go beyond the farthest lossy link. Consequently, request messages would travel long paths in all directions, resulting in high overhead. Therefore, we do not attempt to have all member’s requests attempt to reach beyond the lossy link. In fact, to minimize the hop limit for request messages, our design takes the approach that a member p’s request should extend just far enough to reach some other member q who is closer to the source. If the loss occurred between q and p, then q will be able to retransmit the lost packet. If the loss occurred elsewhere so that q missed the packet as well, then we only need to make sure that q will send a request further up towards the source. Since request timers are set proportional to the measured propagation delays to the source, a member can assume that it is the closest one behind the lossy link if its request timer expires first. It does not matter if a further away member sends an unsuppressed request which does not reach beyond the lossy link; all that matters is that at least one request makes across the lossy link, and it is likely that the request comes from the closest member behind the lossy link. Other members simply rely on this request to trigger the retransmission of the lost data.

Note that a request is used to suppress other duplicate requests as well as to ask for repair. While limiting the hop count of the request message limits the overhead it generates, it also diminishes its ability to suppress the same requests from other members. However, in general, the request hop count in our mechanism is relatively small in comparison with the distance to the source. As a result, the request traffic per loss is still acceptable even though multiple requests of the same retransmission are presented. Moreover, because request timers are based on measured distances to the source, a member far behind the lossy link may receive a reply before sending a request, which relaxes SRM’s previous suppression requirement that a request should reach all members who share the same loss.

On the other hand, we do require that a reply reach all members who suffer the same loss. Since a replier does not know where a packet is dropped, it is difficult for the replier to identify who shares the loss. Because a requester assumes it is immediately behind the lossy link (members closer to the requester are likely to be the ones whose replies are not suppressed), it can determine (as we show below) an upper bound on the hop count needed to reach all other members who share the
loss. The upper bound is called the proxy hop count because the requester acts as proxy for members who share the same loss. The replier determines its reply hop count based on incoming requests. For example, if a replier receives a request from a requester $h$ hops away, and the proxy hop count of the requester is $\rho$, then the replier's reply hop count, $\Pi$, is given by $\Pi = h + \rho$. Note that the reply hop count is an upper bound and the reply may reach members who do not share the loss.

Our hop-scoped error recovery requires a member to measure its distances, in terms of the number of hops, to other members in a session. The distance is measured by exchanging session messages. Since session messages are periodic, the measurement is also periodically refreshed. We will discuss the algorithm to determine the request and proxy hop counts in Section 3.1 and Section 3.2. The detailed mechanism is described in Section 3.3.

### 3.1 Request Hop Count

Since each requester sets its request timer according to its measured distance to the data source, a requester near a lossy link should detect a loss and send its request first. Its request is most efficient in terms of both the recovery delay and the request scope. Based on this observation, a requester can rely on other requesters closer to the lossy link to ask for repair. For example, in Figure 2, a packet from source $s$ is dropped downstream of member $r$. Member $p$ is the closest requester to the lossy link. Member $q$ does not have to send its request to reach $r$ because the request from $p$ can request retransmission for it. Note that, instead of trying to explicitly identify the lossy link, each requester assumes itself the immediate member downstream of the lossy link and sets its request hop count large enough to reach at least one member beyond the lossy link \(^4\). Even though its request may not reach a member with the requested data for a particular loss, the requester can

\(^4\)The member beyond the lossy link can be either an upstream member or a sibling to the requester.
rely on other requesters that are upstream whose request will reach a member with the requested data. Hence, the request hop count, \( \pi_p^s \), for a member \( p \), regarding a source \( s \), in a session \( G \), can be set to,

\[
\pi_p^s = \min\{h_{pq} \mid \forall q \in G, h_{sq} < h_{sp}\}
\]

\( h_{pq} \) is the distance, in terms of the number of hops, between \( p \) and \( q \). \( h_{pq} \) is determined using distance information obtained from session message exchange. Note that the request hop count is calculated on a per-source basis under the assumption that source-specific distribution trees are used.

### 3.2 Proxy Hop Count

A requester sets its proxy hop count so as to reach other members that share the same loss. Since a requester has no knowledge of the underlying network topology, it can only estimate its proxy hop count.

A requester only has to consider members farther away from the source in determining its proxy hop count. There are four kinds of relationship between a requester and a member farther away from the source. They are demonstrated in Figure 2 by member pairs \( \{p,q\}, \{p,u\}, \{p,v\} \) and \( \{p,w\} \).

\( p \) and \( q \) have an upstream-downstream relationship. \( q \) most likely shares losses with \( p \). An upstream member should be proxy for its downstream members to request retransmission. If \( q \) is downstream of \( p \), then \( h_{sp} + h_{pq} \) will be equal to \( h_{sq} \). However, because a member may be one hop away from its first-hop router, \( h_{sp} + h_{pq} \) may be two hops greater than \( h_{sq} \). We refer to the downstream distance of \( p \) regarding a source \( s \) as \( \rho_{1_p}^s \).

\[
\rho_{1_p}^s = \max\{h_{pq} \mid \forall q \in G_p^s, h_{sp} + h_{pq} \leq h_{sq} + 2\}
\]

\( G_p^s \) is the set of members who are farther away from \( s \) than \( p \) is.

\( p \) and \( u \) are siblings and \( p \) is within \( u \)'s request hop count, i.e., \( u \) requests \( p \) for repair. \( p \) has to be proxy for \( u \) as well as \( u \)'s downstream members. We refer to this distance as \( \rho_{2_p}^s \).

\[
\rho_{2_p}^s = \max\{h_{pu} + \max\{\rho_{1_u}^s, \rho_{2_u}^s\} \mid \forall u \in G_p^s, h_{pu} \leq \pi_u^s\}
\]

\( p \) and \( v \) are also siblings and \( v \) is within \( p \)'s request hop count, but \( p \) is not within \( v \)'s request hop count. \( p \) has to be proxy for \( v \) and \( v \)'s downstream members because \( v \)'s requests may be suppressed by \( p \). We refer to this distance as \( \rho_{3_p}^s \).

\[
\rho_{3_p}^s = \max\{h_{pv} + \max\{\rho_{1_v}^s, \rho_{2_v}^s\} \mid \forall v \in G_p^s, h_{pv} \leq \pi_v^s\}
\]

Note that, since \( v \) is suppressed by \( p \), \( p \) does not have to consider \( \rho_{3_v}^s \).

\( p \) and \( w \) are also siblings but they are not within the request hop counts of each other, i.e., \( w \) does not ask \( p \) for repair and its requests are not suppressed by \( p \). Therefore, \( w \) must send its own requests and \( p \) need not be proxy for \( w \).

A requester can determine its proxy hop count, \( \rho \), by taking the maximum of its \( \rho_{1} \), \( \rho_{2} \) and \( \rho_{3} \) from all members from which it hears session messages. Note that the proxy hop count must also be calculated on a per-source basis.

---

1. Issues related to the use of other types of distribution trees are left for future study.
3.3 Mechanism Description

As described above, members in a session exchange session messages to measure the distance, in terms of the number of hops, to other members, and the distances between each pair of members are used to compute the request and proxy hop counts. Moreover, proxy hop count information is added to each session message. In particular, for each source \( s \), a member \( p \) includes \( h_{sp}, \pi^+_p \) and \( \max(\rho_1^p, \rho_2^p) \) in its session messages.

The computation of request and proxy hop counts is performed iteratively. A member recomputes its request and proxy hop counts when a session message is received. Since session messages are multicast periodically, it takes approximately one session cycle time \(^6\) to complete the computation of the request hop count. However, the computation of the proxy hop count depends on results from other members, and so it may take several session cycles to converge. Note that, in order to capture session dynamics, the computation of request and proxy hop counts is timestamped and aged, so obsolete results are timed out.

When a loss is detected, a requester sends a request with its request hop count. Normally, at least one request from members behind the lossy link reaches a member with the requested data and triggers a reply. However, if no reply is received due to packet loss or underestimated request hop count, the requester then sends a second request globally, and correspondingly the reply will be sent globally as well. The request message carries the distance to the source and the proxy hop count. The distance to the source is used to determine request suppression. A member suppresses its scheduled request if the received request is from a member closer to the source. Otherwise, the scheduled request should be sent. The replier uses the proxy hop count to determine the reply hop count.

When a replier responds to a request, a different approach would be that the replier multicasts its reply with a hop count \( h \), where \( h \) is the distance to the original requester. After receiving the reply, the original requester relays the reply to other downstream members with its proxy hop count, \( \rho \) \(^1\). However, this 2-step reply-relaying scheme introduces additional delay in loss recovery, which may cause additional duplicate requests being sent. Furthermore, the scopes of the first reply and the relayed reply overlap with one another, so that members within the overlapping area will receive duplicate replies. Since the distance between a replier and a requester, \( h \), is relatively small in the average case, multicasting a reply with a hop count equal to \( h+\rho \) should not introduce significant overhead in terms of network bandwidth. Therefore, in our hop-scoped error recovery, a replier calculates its reply hop count as the sum of the distance to the requester and the proxy hop count of the requester. If multiple requests are received, the replier takes the maximum of its calculations as the reply hop count.

\(^6\) A session cycle time is the period between two consecutive session messages sent by a member.

Since all members send their session messages at the same rate, a member should receive a session message from each member during a session cycle time.
4 Group-Scoped Error Recovery

Ideally, a request should reach a few neighbors with the requested data to ask for repair, and a reply should reach all members who share the same loss. However, the propagation of error recovery traffic is non-directional in hop-scoped error recovery. The wasted bandwidth is not negligible in some cases, especially in terms of the reply traffic. For example, the reply from member \( r \) in Figure 2 propagates upstream as well as downstream to recover a loss. In this section, we consider the use of separate multicast groups to more precisely control the scope of error recovery traffic.

We use the concept of *local recovery groups*. A local group consists of a set of members who share the same losses to at least some degree. Members share the same losses because they share one or more lossy links along the data delivery path from a source. Because we assume source-specific distribution trees, the creation of local groups is on a per-source basis. However, our mechanism does not limit members to a single local group per source. Multiple local groups can be associated with a source where each group is responsible for error recovery of one or more lossy links. For a specific source, the relationship among these lossy links is either ancestor-descendant or siblings; so membership of these local groups are either perfectly nested or disjoint.

Our group-scoped error recovery follows the SRM approach in which each member is autonomous. That is, each member joins or leaves a local group based on its independent decision. There is no centralized coordination among members. Members in a session start with global error recovery. A member who suffers data losses from a source proposes in its request (sent globally) the creation of a local multicast group for error recovery. The creation of the local group is granted by a replier in its reply. Since the reply is sent globally, members who share the same loss join the local group when they receive the reply. These members are called the *normal members* \(^7\) of the group and their subsequent requests are sent to the local group to ask for repair. If a normal member has joined multiple nested groups, it assumes the loss is within the scope of its innermost local group and therefore the member sends requests to the innermost group. If the loss actually occurred in an outer group, another normal member who sees the outer group as its innermost group would have detected the loss and requested for retransmission.

A normal member in a local group measures the extent to which it shares losses with that group. It stays in the group if the degree of loss sharing is high, otherwise it leaves the group. We use the concept of an *error fingerprint* to measure loss sharing. An error fingerprint is the sequence numbers of the last \( f \) losses in the local group, and it is used by session members to determine the degree of loss sharing.

Other members selectively join a local group to help error recovery. They are called *helpers* of the group. When a helper receives a request from a member of a local group, the helper sends its reply to that same local group.

Our group-scoped error recovery follows the “soft-state” approach. That is, membership solicitation and loss sharing measurement are periodically refreshed to capture session dynamics. The mechanism is described in detail in the following

\(^7\)In the rest of the paper, we will use the terms normal member and member interchangeably.
sections. In particular, we discuss the criteria for proposing, granting, joining and leaving local groups.

4.1 Proposing a Local Group

A member proposes a local group if its error rate exceeds $\delta \%$. In the extreme case, we can choose $\delta = 0$ to encourage all error recovery to be handled by local groups. If a member decides to propose a local group, it should wait a period of time before proposing in order to learn of existing local groups. If there is an existing local group, the member should join the existing local group instead of proposing a new one. The details of a member joining an existing local group are discussed in Section 4.7. The waiting period can be measured in terms of time, number of losses or number of received data packets. If the waiting period is long, a member has more chance to learn of existing local groups, so the overhead of unnecessary group creation is reduced. On the other hand, if the waiting period is short, a new group can be created quickly and the overhead of global error recovery is reduced, at the expense of group creation overhead.

A member proposes a new local group in its request message. It includes the proposed multicast group address and the error fingerprint in the request message. Since the proposed local group is not yet created, the member will use the sequence numbers of its own losses as the initial error fingerprint. The request proposing a local group is multicast globally to suppress other group proposals. If a member has joined any local group, it will not propose creation of additional local groups. However, it may join other local groups as appropriate.

4.2 Granting a Local Group

A replier grants the creation of a new local group in its reply. It includes the address and the error fingerprint of the granted local group. The reply granting the new local group is multicast globally to solicit members who share the same loss. Furthermore, the replier joins the local group as a helper. Therefore, at the beginning of group creation there is at least one helper in the group.

4.3 Joining a Local Group as a Normal Member

A member joins a local group if it shares more than $x\%$ of the losses with the group. When a reply granting a new local group is received, a member joins the group if the similarity of its own losses and the error fingerprint of the granted group exceeds $x\%$.

A member can join multiple local groups and these groups are nested. That is, the membership of an inner group is a subset of the membership of an outer group. It is important for all members to maintain a consistent view of group order so they can exercise these nested groups in the same fashion and produce correct loss sharing measurement. Furthermore, the group order is used in error recovery since a member always sends its requests to its innermost local group first. One simple way to determine the order of a local group is by the sequence number of the reply
granting the local group. The sequence number of the reply granting a local group is called the order number of the local group. Generally speaking, a local group granted later has a larger group order number and a larger scope. Note that the session original group is always the outermost group even though it does not have an order number.

The order of nested groups may not reflect their physical scopes at a particular point of time but abnormal cases will be fixed after the requests and replies disseminate completely. For example, in Figure 3, a new member $p$ may propose a new group $G_2$ before learning of the existing local groups, $G_1$ (Figure 3.a and 3.b). $p$ will be solicited to join $G_1$ later and then it will use $G_1$ as the innermost local group (Figure 3.c). The membership solicitation scheme is discussed in Section 4.7. At this point of time, the physical scope of $G_1$ is larger than the physical scope of $G_2$. Eventually, $G_2$ will be timed out and disappear (Figure 3.d). The group timeout scheme is discussed in Section 4.6.

The threshold $x$ defines the tradeoffs between the number of nested local groups and the error recovery performance. For larger $x$, more nested local groups are created, and each group has a higher loss sharing ratio and achieves greater efficiency for retransmission. As a result, the group maintenance overhead is higher and the error recovery performance is better. On the other hand, for smaller $x$, fewer nested local groups are maintained and the loss sharing ratio in each local group is lower. In the extreme case, if we choose $x = 0$, there is only one local group in the session to recover all losses; if we choose $x = 100\%$, the number of local groups is equal to the number of lossy links and each local group recovers the losses of a single lossy link.

4.4 Error Recovery in a Local Group

If a loss is detected, a member sends its request to its innermost group on its first try. If there is no reply, it will expand its request scope by trying its next outer group until the loss is recovered. As described earlier, even if a request addressed to the innermost group does not reach a helper, members in the outer group should

---

8To be precise, an order number consists of the sequence number of the reply granting the local group in the high order portion and the local group address in the low order portion.
have detected the loss and sent their requests. As a result, a member addressing its request to its innermost group may receive a reply from its outer group. Therefore, the majority of the losses are recovered on the first try and sending requests to an outer group should happen rarely. Note that, since members in the inner group may rely on members in the outer group to ask for data repair, a member \( p \)'s scheduled request should not be suppressed by a request from a local group \( G \) if \( p \) is not a normal member in \( G \). In other words, a request addressed to a local group should only suppress requests of normal members in the group.

The order number of the addressed group is included in the request message. It is used by a replier to determine the destination group for that reply. A replier sends its reply to the local group to which the request was sent. If there are more than one requests, and they are received from different groups, the replier addresses its reply to the group with the largest order number. Moreover, if a request is addressed to an outer group, the group address, order number and error fingerprint of the inner group are carried in the request message to solicit new members.

4.5 Leaving a Local Group

A normal member measures the degree of loss sharing in each local group it joins. The degree of loss sharing in a local group is the ratio of the number of recovered losses and the number of received replies in the group. For example, the loss sharing can be measured every \( m \) replies received in a local group. To prevent oscillation, exponentially-weighted moving average is adopted. If a normal member’s ratio is smaller than \( x \)%, it leaves the local group.

A helper leaves a local group if it does not act as an active replier in the group. For example, a helper leaves a local group if its last \( k \) consecutive scheduled replies for the local group are suppressed. As a result, there are at most \( k \) helpers in a local group.

4.6 Timing Out a Local Group

If there is no error recovery traffic in a local group, the local group should be timed out to reduce group maintenance overhead. Both helpers and members determine when a local group is dormant and leave the group. The time out period can be measured in terms of seconds or the number of received data packets.

4.7 Soliciting New Members

Since an error fingerprint is a snapshot of the group losses, a member who shares the majority of losses with a local group may unfortunately decide not to join the group when it compares its losses with the group error fingerprint. Furthermore, when a new member joins an ongoing session, it has no knowledge of the existing local groups. Therefore, to capture new members as well as old members whose snapshots happened to be skewed, a scheme to periodically solicit new members is required.

\( ^9 \) \( p \) is a helper in \( G \) to be able to receive a request addressed to \( G \).
A local group solicits new members by periodic polling. Members periodically send their requests to the next outer group to solicit new members. Members in the outer group join the inner group based on the comparison of their own losses and the inner group error fingerprint. Since the requests soliciting new members are sent to the next outer group, a new member joins local groups one at a time in an outside-in fashion until it has joined all nested local groups. Note that, if a requester does not receive a reply in its first try, the next request addressed to an outer group can also serve the purpose of membership solicitation. The requests soliciting new members suppress one another to minimize the number of requests addressed to the outer group.

The same scheme is used to solicit new helpers. A replier to a request soliciting new members joins the inner group as a helper. However, a helper that is already in the inner group is closer to the requester and is most likely to respond to the request soliciting new helpers. Therefore, unless all helpers in the inner group have left the session, a new helper will only rarely need to join the inner group.

5 Simulation Results and Discussions

We believe the behavior of our proposed mechanisms might be best understood by first testing a variety of extreme settings before moving on to more complicated scenarios. Consequently, we initially explored our local recovery mechanisms in three extreme but simple topologies – star, string and binary tree – each with a single data source. The star topology represents a session where all members have independent losses. The string topology represents a session where downstream members share the same losses with their upstream members. The binary tree topology represents a mixture of shared and independent losses in a session.

Each topology is tested with four different session sizes: 8, 16, 32 and 64. We simulated the performance of five different mechanisms for each session size: global error recovery, hop-scoped error recovery, and group-scoped error recovery with three different degrees of loss sharing, 33%, 50% and 100%.

We adopted a dynamic timer adjustment mechanism to optimize the performance of the recovery delay, and the number of requests and replies per loss [5]. The general idea of the dynamic timer adjustment mechanism is to make the generation of request and reply timers adaptive to the session environment. A member interprets the feedback from a session as an estimated session size \(^{10}\), and uses the estimated size to tune its request and reply timer parameters, \(A, B, a\) and \(b\). These parameters are described in Section 2 and they determine whether requests and replies are generated aggressively or conservatively. The feedback from the session is the received requests and replies. More duplicate requests and replies imply a larger session size, therefore, a member should increase its timer parameters and send requests and replies conservatively to reduce duplicates. Otherwise, a member should decrease its timer parameters to minimize recovery delay.

\(^{10}\)The estimated session size may not reflect the actual session size. It represents the number of members to compete for sending requests and replies.
Figure 4: Simulation results in star topology: all links with uniformly-distributed error rate

The first set of simulations assumed that all links are with uniformly-distributed error rates and their error rates are fixed throughout the simulation. The result is shown in Figures 4, 5 and 6. The request traffic is the product of the average measured scope a request propagates and the average measured number of requests per loss. The measured request scope is a fraction of the global scope. For example, in global error recovery, the request scope is equal to the global scope since each request is multicast to the entire session, therefore, the request traffic is the average number of request messages per loss. The reply traffic is the product of the average measured scope a reply propagates and the average measured number of replies per loss. The measured reply scope is a fraction of the global scope. The delay ratio is the average ratio of the measured recovery delay and the measured propagation delay from the source.

In the star topology, the distances between each pair of members are equal. Therefore, hop-scoped local recovery performs exactly like global error recovery. Since members have independent losses, there is no loss shared among members. Approximately one request message per loss is generated (Figure 4.a). However, in
Table 1: The number of requests per loss and the number of hops that a request travels in hop-scope error recovery

<table>
<thead>
<tr>
<th>topology</th>
<th>star</th>
<th>string</th>
<th>tree</th>
</tr>
</thead>
<tbody>
<tr>
<td>session size</td>
<td>8</td>
<td>16</td>
<td>32</td>
</tr>
<tr>
<td>request per loss</td>
<td>0.99</td>
<td>0.99</td>
<td>0.99</td>
</tr>
<tr>
<td>request hops</td>
<td>16.0</td>
<td>32.0</td>
<td>64.0</td>
</tr>
<tr>
<td>request traffic</td>
<td>98.7%</td>
<td>98.9%</td>
<td>99.3%</td>
</tr>
</tbody>
</table>

Group-scoped error recovery, each member creates its own local group, so the request and reply traffic is reduced significantly. Note that, because of the constant number of helpers in a local group, the request and reply traffic goes down with the session size (Figure 4.a and 4.b). On the other hand, since there is little loss sharing, the number of available helpers for a specific loss is large in global error recovery and hop-scoped error recovery. As a result, there are multiple replies generated per loss and the reply traffic increases with the session size (Figure 4.b).

The number of local groups in group-scoped error recovery is equal to the number of lossy links. In general, if there are \( n \) members in a session, the number of local groups is equal to \( n \). Each local group recovers \( \frac{1}{n} \) of total losses in the session and its scope is roughly \( \frac{1}{n} \) of the session scope. Therefore, we can estimate a lower bound on the request and reply traffic in group-scoped error recovery as \( \frac{1}{n} \) of the traffic in global error recovery. Since this estimated request and reply traffic is a lower bound, it represents the greatest degree of savings possible.\(^{11}\) The estimated values are shown as gray curves in Figure 4.

In the string topology, downstream members share all losses with their upstream members. A downstream member can rely on its upstream members to ask for repair in hop-scoped error recovery. As a result, request traffic in hop-scoped error recovery is reduced significantly. Moreover, request traffic in hop-scoped error recovery goes down with the session size because more downstream members rely on their upstream members to ask for repair.

Table 1 shows the average number of requests per loss and the average number of hops that a request message travels in hop-scoped error recovery.\(^{12}\) The number of requests per loss in the string topology increases with the session size and the number of hops that a request message travels remains constant. However, the increase in the number of requests per loss is sub-linear in terms of the session size; therefore, the average request traffic still decreases with the session size. As described earlier, even if multiple requests per loss are presented in hop-scoped error recovery, the overall request traffic is improved because the scope of each request is small and a member far behind a lossy link may receive a reply before even sending a request.

On the other hand, in group-scoped error recovery, request messages propagate

\(^{11}\)Upper bound estimates would need to take several other factors into account. For example, the number of helpers and membership dynamics in a local group.

\(^{12}\)The number of requests per loss in the star topology is less than one because some of the requests are queued in the network when the simulation terminates.
to all downstream members, so request traffic goes up with the session size (Figure 5.a). In terms of reply traffic, hop-scoped error recovery performs worse than group-scoped error recovery because hop-scoped error recovery does not regulate the direction in which the reply messages propagate (Figure 5.b). Since request messages only reach a small number of members in hop-scoped error recovery, the estimated session size is much smaller. As a consequence, members send requests more aggressively and the recovery delay is reduced. On the other hand, in group-scoped error recovery, there is only a small number of helpers in a local group, and hence, the recovery delay in group-scoped error recovery is greater than the recovery delay in global error recovery (Figure 5.c).

The tree topology is a mixture of the star and string topologies. The performance of hop-scoped error recovery is similar to group-scoped error recovery with $x = 100\%$ in terms of request traffic, and it is similar to group-scoped error recovery with $x = 33\%$ in terms of reply traffic. Note that both request and reply traffic decreases with the session size in group-scoped error recovery.

The number of local groups in group-scoped error recovery in the string and
Figure 6: Simulation results in tree topology: all links with uniformly-distributed error rate

tree topologies is proportional to \( x \). For example, if \( x = 100\% \), each local group is responsible for the error recovery of a single lossy link. The number of local groups is equal to the number of lossy links in the session. If \( x = 50\% \), a local group is responsible for the error recovery of two lossy links and the number of local groups is equal to half of the number of lossy links. In general, for a session of size \( n \), the number of local groups is \( x \cdot n \), the number of lossy links covered by a local group is \( \frac{1}{x} \), and the percentage of losses recovered by a local group is \( \frac{1}{x} \). Therefore, the estimated error recovery traffic, \( T \), can be calculated as \( T\% = \frac{1}{x} \cdot \sum_{i=1}^{x} \frac{f(i)}{n} \), where \( f(i) \) is the size of the \( i \)th local group. For string topology, \( f(i) = n - \frac{i-1}{x} \). For tree topology, \( f(i) \approx 2^{\log_2{n} - \log_2{(\frac{i+1}{x})}} \). Therefore, the estimated lower bounds of the error recovery traffic in string and tree topologies are,

\[
T_{\text{string}}\% = \frac{x \cdot n + 1}{2 \cdot x \cdot n} \quad \quad T_{\text{tree}}\% = \sum_{i=1}^{x} \frac{1}{(x + i - 1) \cdot n}
\]

The estimated values are shown as gray curves in Figures 5 and 6.

In the second set of simulations, a randomly selected 12.5\% of the links are
<table>
<thead>
<tr>
<th>%</th>
<th>source s1</th>
<th>s2</th>
<th>s3</th>
<th>p3</th>
<th>r3</th>
<th>s4</th>
<th>p2</th>
<th>p1</th>
<th>q2</th>
<th>q1</th>
<th>r1</th>
<th>r7</th>
<th>r6</th>
<th>r5</th>
<th>r4</th>
<th>r2</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>100.0</td>
<td>99.5</td>
<td>99.1</td>
<td>98.9</td>
<td>99.9</td>
<td>99.3</td>
<td>98.5</td>
<td>99.0</td>
<td>99.1</td>
<td>98.9</td>
<td>99.3</td>
<td>99.0</td>
<td>98.9</td>
<td>99.0</td>
<td>99.0</td>
<td>99.0</td>
</tr>
<tr>
<td>16</td>
<td>100.0</td>
<td>99.5</td>
<td>99.1</td>
<td>98.9</td>
<td>99.9</td>
<td>99.3</td>
<td>98.5</td>
<td>99.0</td>
<td>99.1</td>
<td>98.9</td>
<td>99.3</td>
<td>99.0</td>
<td>98.9</td>
<td>99.0</td>
<td>99.0</td>
<td>99.0</td>
</tr>
<tr>
<td>32</td>
<td>100.0</td>
<td>99.5</td>
<td>99.1</td>
<td>98.9</td>
<td>99.9</td>
<td>99.3</td>
<td>98.5</td>
<td>99.0</td>
<td>99.1</td>
<td>98.9</td>
<td>99.3</td>
<td>99.0</td>
<td>98.9</td>
<td>99.0</td>
<td>99.0</td>
<td>99.0</td>
</tr>
<tr>
<td>64</td>
<td>100.0</td>
<td>99.5</td>
<td>99.1</td>
<td>98.9</td>
<td>99.9</td>
<td>99.3</td>
<td>98.5</td>
<td>99.0</td>
<td>99.1</td>
<td>98.9</td>
<td>99.3</td>
<td>99.0</td>
<td>98.9</td>
<td>99.0</td>
<td>99.0</td>
<td>99.0</td>
</tr>
</tbody>
</table>

Table 2: Simulation results of 12.5% links with uniformly-distributed error rate

Figure 7: Mbone-like topology used in our simulations

with uniformly-distributed error rates and their error rates are fixed throughout the simulation. The results are shown in Table 2. Both hop-scoped and group-scoped error recoveries outperform the global error recovery in terms of the request and reply traffic, except for hop-scoped error recovery in the star topology. In the string topology of size 16, the reply traffic in hop-scoped error recovery is close to 100% because the randomly-selected lossy links are in the middle of the string topology. Since hop-scoped error recovery does not regulate the traffic direction, the reply traffic propagates to the entire session membership. The same scenario can be applied to the reply traffic in the tree topology. In general, hop-scoped error recovery performs worse than group-scoped error recovery in terms of the reply traffic if the lossy links are sparsely distributed.

The local error recovery mechanisms were also simulated in a Mbone-like topology shown in Figure 7. Nodes connected with thick lines symbolize the Mbone. Other nodes represent local area networks. Session members are represented by black nodes and one of them, s1, is selected as the data source. The lossy links are
Figure 8: Average request and reply scopes of individual members

represented by gray lines. We assume most of the losses are at local area networks.

Figure 8.a and 8.b show the measured request and reply scopes of individual members as well as the average request and reply scopes during the simulation. The scope is measured in terms of the number of hops that requests and replies travel. The average request and reply scopes are consistent with the results from the star, string and tree topologies. The performance of hop-scoped error recovery is very competitive with group-scoped error recovery in terms of the request scope. However, the reduction of reply scope is limited in hop-scoped error recovery.

In Figure 8.a, the request scope in group-scoped error recovery goes down as x goes up. A large x means higher loss sharing, therefore, there are fewer members in a local group and they have a higher degree of loss sharing. As a consequence, the requests and replies in the group are required by more members in the group and less bandwidth is wasted. Most members have relatively small request scopes in both hop-scoped and group-scoped error recoveries. This means that most of the requests are sent only within the local area networks. Note that member r_1 has a relatively large request scope in both hop-scoped and group-scoped error recoveries because its requests has to propagate across the Mbone to reach a helper.

The reply scope, shown in Figure 8.b, depends on where the request is coming from. A small reply scope means the reply is within local area networks. For example, r_1’s reply scope in group-scoped error recovery is small because r_1 is only responsible to recover losses of its downstream members within its local area net-
work. However, \( r_1 \)'s reply scope in hop-scoped error recovery almost doubles its reply scope in group-scoped error recovery because hop-scoped error recovery does not regulate the direction of reply traffic. On the other hand, a large reply scope means the reply is multicast across the Mbone. For example, member \( s_2, s_3, s_4, p_1, p_2 \) and \( p_3 \) have relatively large reply scopes in both hop-scoped and group-scoped error recoveries, which means their replies respond to requests from remote members across the Mbone. However, response to remote requesters happens rarely. As seen in Figure 8.b, the average reply scope in both hop-scoped and group-scoped error recoveries are still smaller than the reply scope in global error recovery.

Figure 9.a and 9.b show the measured request traffic and reply traffic during the simulation. The request and reply traffic is measured in terms of the number of hops that requests and replies travel. The convergence periods of request and reply scopes are short in both hop-scoped and group-scoped error recoveries. Generally speaking, the convergence time of request scope in hop-scoped error recovery is approximately one session cycle time. The measurement of reply scope in hop-scoped error recovery relies on the results from other members, so the convergence could take several session cycles. The convergence time in group-scoped error recovery depends on the number of nested local groups and their order of creation. If the innermost group is created first, the convergence period is shorter. On the other hand, if the group with the largest scope is created first, this group has to be shrunk before the second nested group can be created. Therefore, it takes a longer time to converge. For
example, if there are \( n \) nested groups. The average waiting time to propose a group creation is \( t_1 \) and the average shrinking time is \( t_2 \). In the worst case, the convergence time is approximately \( n \cdot (t_1 + t_2) \). The shrinking time is a function of the period of a loss sharing measurement.

To further understand the behavior of convergence time of a new member joining an ongoing session, we manually reset the state of member \( r_7 \) during the next simulation. The measured request traffic of \( r_7 \) during the simulation is shown in Figure 10. \( r_7 \) starts with global error recovery after its state is reset at simulation time 300 seconds. However, \( r_7 \) will calculate its request hop count based on the incoming session messages in hop-scoped error recovery or learn of the existing local groups in group-scoped error recovery to restore its state, therefore, its request traffic is reduced. Note that the request traffic in group-scoped error recovery oscillates because the request scope, the scope of the local group to which requests are sent, is dynamically adjusted based on observed loss patterns. On the other hand, the request traffic in hop-scoped error recovery is fairly stable as long as the topology remains unchanged.

From the simulations, we found that group-scoped error recovery performs better than hop-scoped error recovery in terms of the reply traffic. However, hop-scoped error recovery performs better than group-scoped error recovery in terms of the request traffic, except in the star topology. Since the size of a request message is much smaller than the size of a reply message, it is more important to reduce the reply traffic than to reduce the request traffic. Therefore, in terms of the traffic reduction, group-scoped error recovery appears to provide a better solution. On the other hand, if we consider other sources of overhead introduced by these two approaches, it appears that group-scoped error recovery imposes more overhead on session members\(^\text{13}\) as well as the underlying multicast routing.

\(^\text{13}\) Group-scoped error recovery requires the host to send periodic IGMP messages to refresh the multicast delivery path for each local group.
There have been several other treatments of error recovery for reliable multicast transport. In contrast to our proposal which assumes session members are autonomous, these previous works require various degrees of static configuration or centralized coordination.

Hofmann [6, 7] proposed a “local group concept”. A session is split into subgroups and each subgroup combines members in a local region. A subgroup is represented by a local group controller which supports local loss retransmission. The establishment of local groups is supported by a communication service, named Group Distance Service. A member searches and joins the closest local group. If no suitable group exists, the member will establish a new local group and appoint itself as the controller.

Kasera et al. [8] examined the approach of using separate multicast groups to recover individual losses in reliable multicast communication. Lost packets are categorized into groups, the retransmission of a lost packet is multicast to the group it belongs to. Receivers dynamically join and leave those groups to recover packet losses.

TMTP [9] groups session members into domains and organizes these domains into a hierarchic control tree to improve the scalability of error recovery. Members in a domain request the domain manager for retransmission. A domain manager is also responsible for error recovery of its children managers in the control tree. The scope of retransmission is restricted by using the TTL field. The control tree is self-organized, and it is built dynamically as domain managers join and leave the session.

Holbrook et al. [10] suggested a hierarchic logging server structure to reduce error recovery traffic in a multicast session. The distribution and hierarchy of logging servers is statically configured. Receivers contact their local secondary server for retransmission instead of the remote primary servers to avoid NAK implosion, and to minimize recovery latency and bandwidth. A server either unicasts or multicasts a retransmission based on the number of requests it receives.

RMTP [11, 12] adopts a similar hierarchic structure to avoid message implosion. A set of designated receivers (DR) is selected statically in a session. DRs are capable of retransmitting lost data. However, the hierarchy of DRs is constructed dynamically. Each receiver selects its least upstream DR as the ACK processor (AP), and periodically sends its receiving state to the AP to request retransmissions. A retransmission is either unicast or multicast based on the number of requests.

7 Conclusion
We proposed two different approaches to reduce error recovery traffic in SRM. In hop-scoped error recovery, members calculate the required hop counts for their requests and replies based on distance information exchanged in session messages. Since the information is piggybacked on their session messages, the overhead imposed by hop-scoped error recovery is relatively small. However, hop-scoped error
recovery does not regulate traffic direction. If the topology of a session is star-shaped, hop-scoped error recovery does not perform much better than global error recovery.

Group-scoped error recovery bounds the scope of error recovery traffic by using separate multicast groups. Members that share the same losses join a local group for error recovery, thus the error recovery traffic is only distributed within the local group. Group-scoped error recovery requires individual members to maintain multiple local groups. Therefore, more overhead is imposed on members as well as on the underlying multicast routing.

There remain several open issues. In hop-scoped error recovery, maintaining a pair of request and reply hop counts for individual sources does not introduce significant overhead. However, maintaining multiple local groups for individual sources in group-scoped error recovery may not be acceptable. Further research should look into group aggregation across sources. A local group is associated with one or more lossy links. Sources who share the same delivery path and the same lossy links along the path should be considered the same in terms of error recovery. Therefore, error recovery of losses from these sources should be handled by a single local group.

Another scenario that we have not fully understood is the convergence time of group-scoped error recovery in the presence of network dynamics. For example, if the network topology changes, members in a local group do not share losses anymore. Therefore, members have to readjust themselves so that the new membership in the local group represents a set of members who share the same losses. Another example of network dynamics is traffic congestion. Data losses due to congestion changes the error rates and the locations of lossy links in a session. Since local groups are associated with lossy links, changes in error rates and locations of lossy links affect the loss sharing behavior within local groups. The membership in a local group has to be readjusted in order to adapt to these changes. The study of the convergence time of membership readjustment can help us to better understand the tolerance to network dynamics of group-scoped error recovery.

Finally, one might consider combining these two approaches by using a hop-scoped on the request messages sent to local groups since hop-scoped error recovery produces better traffic reduction in terms of the request traffic. In our hop-scoped scheme requests are addressed to the global session group with a specific hop count and that hop count is determined by exchanging session messages and measuring how far is the closest upstream neighbor. However, if a hop-scoped request is sent to a local group, it can only guarantee a response if the requester knows both how to set the hop count and how to choose the appropriate local group. Our hop-scoped error recovery only provides the former information. The requester would analyze session messages, determine an appropriate hop count, but then the target upstream neighbor might not be a member of that local group. More research has to be done to ensure that either the closest upstream neighbor joins the same local group or the requester only considers members in the same local group in computing its request hop count.
References


