Local Error Recovery in SRM : Comparison of Two Approaches

Ching-Gung Liu, Deborah Estrin, Scott Shenker and Lixia Zhang

***Abstract*— SRM is a 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. While SRM effectively uses random timers to suppress duplicate requests and replies, the global nature of the request and replies means that every packet loss results in at least one request and reply message sent to the entire group.**

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 distribution of requests and replies, whereas the second approach confines error recovery traffic using separately-addressed 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

Scalable Reliable Multicast (SRM) [1, 2] is a framework for reliable multicast delivery; it guarantees data delivery to all members in a multicast session [3]. The mechanisms needed to achieve this reliability can be decomposed into two parts: session message exchange 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, so that losses can be detected) and to determine the propagation delays between each pair of members. [1] The error recovery mechanism is receiver-initiated and NAK-based [4]; receivers are responsible for detecting data losses and requesting retransmissions. These retransmission requests, and the resulting replies, are multicast to the entire group. Members use propagation delays to schedule their request and reply timers; each member detecting a loss waits for a random time period before sending the retransmission request, and similarly each member receiving a retransmission request waits for a random time period before sending the reply. [2] When members receive a retransmission request (or, respectively, a reply message) while waiting to send one of their own, they cancel their scheduled transmission. This enables SRM to suppress duplicate requests and replies, and thus avoid the request and reply message implosion problem [5].

However, each packet loss will result in at least one request and one reply message being sent to the entire multicast group. This limits the scalability of SRM as network and group size increases [6, 7]. As suggested in [2], the premise of this paper is that the error recovery mechanism should isolate error recovery traffic to the required scope. 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 by distributing it to subsets of the group using separate multicast addresses. Simulation results of these mechanisms suggest that they both reduce error recovery traffic without introducing significant overhead. [3]

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. In both mechanisms that we propose here, a member may occasionally send its requests and replies to an inappropriate scope. While such "mistakes" have a slight impact on the performance of SRM (in terms of the volume of error recovery traffic), they have no impact on its correctness, since all data losses are eventually recovered.

The paper is organized as follows: Sections 2 and 3 describe the hop-scoped error recovery and the group-scoped error recovery mechanisms, respectively. Section 4 presents the simulation models and analyzes the simulation results, and Section 5 reviews related work. We conclude in Section 6 with a short summary.

Ching-Gung Liu is with the Fujitsu Laboratories of America, Inc. (e-mail: *charley@fla.fujitsu.com*).
Deborah Estrin is with the Computer Science Department/Information Sciences Institute, University of Southern California (e-mail: *estrin@usc.edu*).
Scott Shenker is with the Xerox Palo Alto Research Center (e-mail: *shenker@parc.xerox.com*).
Lixia Zhang is with the Computer Science Department, University of California, Los Angeles (e-mail: *lixia@cs.ucla.edu*).
This research was supported in part by the Advanced Research Projects Agency, monitored by Fort Huachuca under contracts DABT63-94-C-0073, and by the National Science Foundation under grant award No. NCR-96-28-729. The views expressed here do not reflect the position or policy of the U.S. government.

[1] Although the delay from a member and the delay to a member may be different, SRM assumes the path between a pair of members are symmetric; thus each member determines its one-way delay to another member by taking half of its measured round-trip delay. This symmetry assumption should cause no performance penalty even when the paths are asymmetric, because the delay between members is mainly used to differentiate members in setting their retransmission timers.

[2] SRM assumes most or 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] The discussion of localizing session messages is outside the scope of this paper. More information can be found in [8, 9].

2 Hop-Scoped Error Recovery

The simplest way to control the scope of requests and replies is to limit the number of hops they travel. [4] We wish to use the minimum hop counts possible in requests and replies. To minimize the hop limit for request messages, our design takes the approach that a member p's request extends 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. All that matters is that at least one request makes it across the lossy link (*i.e.*, link where the loss occurs), and it is likely that this request comes from the closest member behind the lossy link since each member's request timer is set proportional to the measured delay from the source.

In the original SRM design a request is used to suppress 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 other members from sending the same requests. Fortunately, the request hop count in our mechanism, generally speaking, is relatively small compared to the distance (in terms of the number of hops) to the source. Thus the request overhead per loss is acceptable even though multiple requests for retransmitting the same data are generated. Moreover, because request timers are based on the propagation delay from the source, a member far behind the lossy link may receive a reply before sending its own request.

While we greatly limit the scope of requests, we require that a reply have sufficient scope to reach all members who share the same loss. Since a replier does not know where a packet is dropped, it is difficult for the replier to decide how far the retransmission must go. However, if a requester assumes it is immediately behind the lossy link, it can determine (as we show below) an upper bound on the hop count needed to reach all other members behind the lossy link. The upper bound is called the *proxy hop count* because the requester acts as request proxy for members who share the same loss. When a replier receives a request from a requester h hops away, and the proxy hop count of the requester is P, then the replier's reply hop count, Π, is given by $\Pi = h + P$. 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 all the other members in the same 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 2.1 and Section 2.2. The detailed mechanism is described in Section 2.3.

[4] In [2], a 2-step reply-relying hop-scoped error recovery mechanism is suggested, but without a specific method to measure the request and reply hop counts. Our proposed mechanism can also be used to measure the request and reply hop counts in [2].

2.1 Request Hop Count

Each requester simply sets its request hop count large enough to reach at least one member that is closer to the source. This member does not necessarily share the same data delivery path with the requester; all that matters is that the upstream member is closer to the source than the requester. Hence the hop count to reach an upstream member for a member p regarding a source s in a session G, ${d_1}_p^s$, can be set to,

$$ {d_1}_p^s = \min\{h_{pq} \mid \forall q \in G, h_{sq} < h_{sp}\} $$

where h_{pq} is the distance, in terms of the number of hops, from p to q.

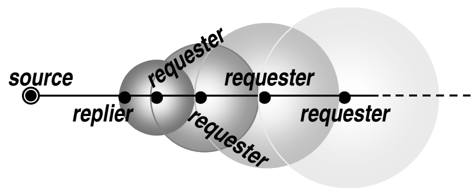

Figure 1: Multicasting requests with a limited hop count reduces the effectiveness of request suppression. (Scenario in the string topology)

Because a limited hop count reduces the effectiveness of request suppression, multiple requests regarding the same loss may be generated. In particular, two scenarios illustrated in Figure 1 and 2 are known to cause duplicate requests. In Figure 1, the hop count to an immediate upstream member is smaller than the hop count to an immediate downstream member. Therefore, requests sent by upstream members can not reach their downstream members to suppress them from sending out the same requests. In the worst case, the request overhead within the loss region can be two requests per link, one traveling upstream and the other downward. We consider such overhead acceptable.

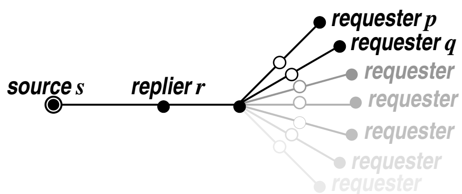

Figure 2: Multicasting requests with a limited hop count reduces the effectiveness of request suppression. (Scenario in the star topology)

The scenario illustrated in Figure 2 can cause much higher request overhead. All the requesters in Figure 2 choose r as the replier. They multicast their requests with a hop count value that is big enough to reach r, but not big enough to reach one another for duplicate suppression. In the worst case the number of requests per loss within the loss region is equal to the number of requesters. Therefore to minimize the number of duplicate requests, requesters have to set their request hop counts large enough not only to reach the replier but also to reach one another for suppression. [5]

The characteristic of this scenario in Figure 2 is that multiple requesters do not choose one another as repliers for retransmission. For example, both p and q request r who is closer to the source than p and q. Because the replier selection is autonomous and independent, in general a member will not know the selection of other members. However, two members p and q are very likely to be in the scenario described in Figure 2 if they fall within the following three conditions:

1. p requests someone closer to the source than q,
2. q requests someone closer to the source than p, and
3. their request scopes overlap with each other.

To check Conditions 1 and 2, a member p needs to calculate the distance from source s to its intended replier (denoted by $d2_p^s$).

$$d2_p^s = \min\{h_{sr} \mid \forall r \in G, h_{pr} \leq d1_p^s\}$$

If $d2_p^s < h_{sq}$, then p concludes that it does not request member q for retransmission. Similarly, if $d2_q^s < h_{sp}$, then p concludes that it is not the intended replier for member q. For Condition 3, p concludes that their request scopes overlap if $(d1_p^s + d1_q^s) > \min\{h_{pq}, h_{qp}\}$, Therefore, member p can compute the required hop count to suppress other requests as $d3_p^s$,

$$d3_p^s = \max\{h_{pq} \mid \forall q \in G,$$

$$(d2_p^s < h_{sq}) \wedge (d2_q^s < h_{sp}) \wedge (d1_p^s + d1_q^s > \min\{h_{pq}, h_{qp}\})\}$$

The request hop count of p regarding source s (denoted by π_p^s) is the maximum of $d1_p^s$ and $d3_p^s$, *i.e.*,

$$\pi_p^s = \max\{d1_p^s, d3_p^s\}$$

The request hop count is calculated on a per-source basis under the assumption that source-specific multicast distribution trees are used. Issues related to the use of other types of multicast trees are left for future study.

2.2 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 an upper bound of its proxy hop count.

A requester only has to consider members farther away from the source than itself in determining its proxy hop count. There are four kinds of relationship between a requester and a member farther from the source. They are demonstrated in Figure 3 by member pairs $\{p,q\}$, $\{p,u\}$, $\{p,v\}$ and $\{p,w\}$.

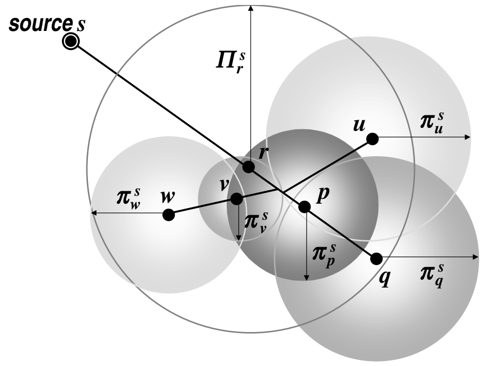

Figure 3: Request and reply hop counts (Thick lines represent data delivery path, circles represent the regions of request and reply scopes.)

1. p and q have an upstream-downstream relationship. That is, The path from s to p is a subset of the path from s to q. In this case 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 $h_{sp} + h_{pq}$ is 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 $D1_p^s$.

$$D1_p^s = \max\{h_{pq} \mid \forall q \in G,$$

$$(h_{sp} < h_{sq}) \wedge (h_{sp} + h_{pq} \leq h_{sq} + 2)\}$$

2. p and u are siblings (one's path from s is not a subset of the other) and u requests p for repair. In other words, p is closer to source s than u and p is within the radius of u's request hop count (To be more specific, p is within the radius of $d1_u^s$.) Therefore, p has to be proxy for u as well as u's downstream members. We refer to this distance as $D2_p^s$.

$$D2_p^s = \max\{h_{pu} + \max\{D1_u^s, D2_u^s\} \mid \forall u \in G,$$

$$(h_{sp} < h_{su}) \wedge (h_{up} \leq d1_u^s)\}$$

3. p and v are also siblings and v is within the radius of p's request hop count ($h_{pv} \leq \pi_p^s$), although v is not p's intended replier ($d2_p^s < h_{sv}$). 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 $D3_p^s$. Since v's requests are suppressed by p, p does not have to consider $D3_v^s$ in computing $D3_p^s$.

$$D3_p^s = \max\{h_{pv} + \max\{D1_v^s, D2_v^s\} \mid \forall v \in G,$$

$$(d2_p^s < h_{sv}) \wedge (h_{pv} \leq \pi_p^s)\}$$

4. 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 sends its own requests and p does not need to proxy for w.

The proxy hop count is calculated on a per-source basis. A requester p can determine its proxy hop count regarding

[5] Sally Floyd helped identify this pathological case in an earlier version of this paper.

source s, P_p^s, by taking the maximum of its ${D_1}_p^s$, ${D_2}_p^s$ and ${D_3}_p^s$ from all members from which it hears session messages.

$$P_p^s = \max\{{D_1}_p^s, {D_2}_p^s, {D_3}_p^s\}$$

A replier r determines its reply hop count triggered by an incoming request from p regarding a loss from s, Π_r^s, as

$$\Pi_r^s = h_{rp} + P_p^s$$

2.3 Mechanism Description and Discussion

Members exchange session messages to measure the distance, in terms of the number of hops, from and to other members. Additional information is carried in each session message. In particular, member p includes h_{sp}, ${d_1}_p^s$, ${d_2}_p^s$ and $\max\{{D_1}_p^s, {D_2}_p^s\}$ for each source s, and h_{pq} for each member q, in its session messages. The distance between each pair of members is used to compute the request and proxy hop counts.

The computation of request and proxy hop counts is performed iteratively. A member recomputes its request and proxy hop counts when a new session message is received. It takes as few as two session cycles [6] for a member to compute its request hop count. The computation of the proxy hop count takes several session cycles to converge because it depends on results from other members (D_2, to be specific). In order to capture session dynamics, the computation of request and proxy hop counts is timestamped and aged, so obsolete results will be timed out.

When a packet loss is detected, a requester multicasts a request within the radius of its request hop count. The request message carries distance from the source to the intended replier (d_2), and the proxy hop count. d_2 is used to determine request suppression. A member suppresses its scheduled request if the received request is intended to reach someone closer to the source. Otherwise, the scheduled request should be sent. The replier uses the proxy hop count to determine the reply 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 sends a second request globally, and the corresponding reply will be sent globally as well.

A different approach to reply scope control would be that, when a replier responds to a request, it 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 members downstream within the radius of its proxy hop count, P [2]. However, this 2-step reply-relaying scheme introduces additional delay in reply propagation, which may cause additional duplicate requests being sent. Furthermore, the scopes of the first reply and the relayed reply overlap, thus members within the overlapped area 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 of $h + P$ should not introduce significantly more overhead in terms of network bandwidth. Therefore, in our hop-scoped error recovery, a replier uses $h + P$ for its reply hop count. If multiple requests are received, the replier takes the maximum P value received in its calculation.

One question regarding the hop-scoped error recovery mechanism is that the hop-scoped packets may prevent multicast pruning in dense-mode multicast routing protocols, such as DVMRP [10, 11] and PIM-DM [12, 13]. As a result, hop-scoped multicast packets become locally broadcast, *i.e.,* they reach all the routers within the hop limit including those that are not on the multicast routing tree. However, in SRM, because each member periodically multicasts global session messages, those session messages causes branches that reach no members to be pruned off. Therefore, hop-scoped request and reply traffic is contained within the multicast tree.

Ideally, a request should reach a few neighbors who have the requested data, and a reply should reach only those members who lost the same packet. However, because hop-scoped multicast traffic radiates in all directions, the bandwidth overhead can be significant in some cases, especially in case of the reply traffic. For example, the reply from member r in Figure 3 propagates upstream as well as downstream to recover a loss. In the next section we consider the use of separate multicast groups to more precisely control the scope of error recovery traffic.

3 Group-Scoped Error Recovery

A *local recovery group* consists of a set of members who share common data 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 multicast 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 for one or more lossy links. For a specific source, the relationship among these lossy links is either ancestor-descendant or siblings, so that these local recovery groups are either perfectly nested or totally disjoint, as shown in Figure 4.

Our group-scoped error recovery follows a basic SRM design principle of each member being an autonomous entity. That is, each member makes its own decisions on whether to join or leave a local group. There is no centralized coordination among members. Members use the *error fingerprints* to measure the degree of loss sharing with a local group. An error fingerprint is the sequence numbers of the last f losses in a local group. For example, a member p shares 50% losses with a local group G if p lost more

[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.

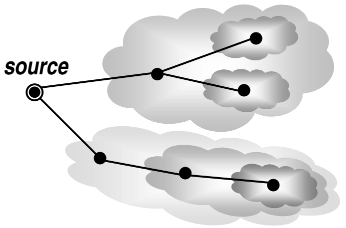

Figure 4: Membership of local recovery groups are either perfectly nested or totally disjoint.

than half of the packets specified in G's error fingerprint.

At the beginning of a session, data losses are recovered globally. A member who suffers noticeable data losses from a source proposes the creation of a local recovery group in its retransmission request (sent globally), together with its error fingerprint. The creation of the local group is granted by a replier in its reply. [7] Since the reply is sent globally, other members who have a matching error fingerprint join the local group when they receive the reply. These members are called the *regular members*, or simply members, of the group and their subsequent requests are sent to the local group. 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.

A regular 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. If a regular member has joined multiple nested groups, it always sends retransmission requests to the innermost group first. In cases the loss actually occurred in an outer group, another member who sees the outer group as its innermost group should have detected the loss and requested retransmission.

Our group-scoped error recovery follows the "soft-state" approach. Membership solicitation and loss-sharing measurement are periodically refreshed to capture session dynamics. The mechanism is described in detail in the following sections. In particular, we discuss the criteria for proposing, granting, joining and leaving local groups.

3.1 Proposing and Granting a Local Group

A member proposes a local group if its error rate exceeds α, where $0 \leq \alpha \leq 100\%$. In the extreme case, we can choose $\alpha = 0$ to encourage all error recovery to be handled by local groups. If a member decides to propose a local group, it waits a period of time before proposing in order to learn of existing local groups. If there is an existing local group, the member joins the existing local group instead of proposing a new one. Section 3.5 discusses in more detail the joining process. The waiting period can be measured in terms of time, number of losses or number of received data packets. A longer waiting period increases the chance for a member to discover existing local groups, thus reduces the overhead of unnecessary group creation. On the other hand, if the waiting period is short, a new group can be created quickly and the overhead of the global error recovery is reduced, at the expense of group creation overhead. Since an error fingerprint is required for proposing a local group, the waiting period before a member proposing a group has to be long enough to cover f losses.

A member proposes a new local group in its request message by including the proposed multicast address and the error fingerprint. Since the proposed local group is not yet created, the member uses the sequence numbers of its own losses as the initial group 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 is not allowed to propose additional local groups. However, it may join other groups as appropriate.

A replier grants the creation of a new local group in its reply. The reply message includes the address and the error fingerprint of the granted local group. This message is multicast globally to solicit members who share the same losses. Furthermore, the replier joins the local group as a helper, which assures that there is at least one helper in the new group.

3.2 Joining a Local Group as a Regular Member

A member joins a local group if it shares more than β of the losses with the group, where $0 \leq \beta \leq 100\%$. 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 β.

If a member joins multiple local groups, these groups must be nested. That is, the membership of an inner group is a subset of the membership of an outer group. It is important that all members maintain a consistent view of group order so they can exercise these nested groups in the same fashion and produce correct loss-sharing measurement. The group order is also used in error recovery process 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. To be precise, an order number consists of the sequence number of the reply in the high order portion and the local group address in the low order portion. We assume a local group address is unique within a session. If multiple local groups are created with the same sequence number, their order numbers are still unique. Generally speaking, a local group granted later has a larger group order number and a larger scope. Note that the original session 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, a transient phenomenon that will be fixed after the requests and replies

[7] A similar approach was proposed in [14, 15] to solve the implosion of multicast congestion feedback.

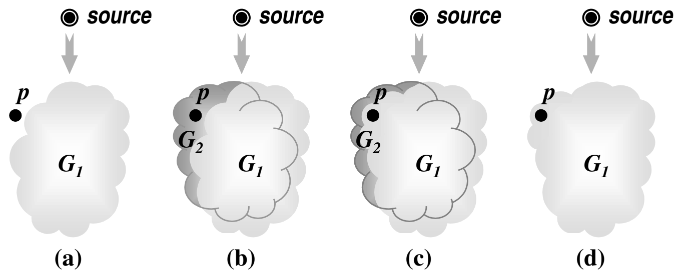

Figure 5: Evolution of misplaced nested local groups

disseminate completely. For example, in Figure 5, a new member p may propose a new group G_2 before learning of the existing local groups, G_1 (Figure 5(a) and 5(b)). p will be solicited to join G_1 later and then it will use G_1 as the innermost local group (Figure 5(c)). At this point of time, the physical scope of G_1 is larger than that of G_2. Eventually, G_2 will be timed out and disappear (Figure 5(d)). The group timeout scheme is discussed in Section 3.4 and the membership solicitation scheme is discussed in Section 3.5.

The threshold β defines the tradeoffs between the number of nested local groups and the error recovery performance. For large β value, 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 small β, fewer nested local groups are maintained but the loss-sharing ratio in each local group is also lower. In the extreme case, if we choose $\beta = 0$, there is only one local group in the session to recover all losses; if we choose $\beta = 100\%$, the number of local groups is equal to the number of lossy links and each group recovers losses over each lossy link.

3.3 Error Recovery in a Local Group

When a loss is detected, a member sends its request to its innermost group first. 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 to the innermost group does not reach a helper, members in the outer group should have detected the loss and sent their requests. Therefore, the majority of the losses are recovered quickly and sending requests to an outer group upon timeout should happen rarely. 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 regular member in G (p can be a helper in G to receive a request sent to G.) In other words, a request addressed to a local group should only suppress requests of other regular members in that 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 the corresponding reply. A replier sends its reply to the local group to which the request was sent. If it receives multiple requests that are sent to different groups, the replier addresses its reply to the group with the largest order number.

3.4 Leaving a Local Group

A member measures the degree of loss sharing in each local group it joins by the ratio of the number of its total losses over the number of received replies from 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 member's loss-sharing ratio is smaller than β, it leaves the local group.

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.

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 timeout period can be measured in terms of seconds or the number of received data packets.

3.5 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 when it learns about the group the first time. Furthermore, when a new member joins an ongoing session, it has no knowledge of the existing local groups. A scheme to periodically solicit new members is necessary to capture new members as well as old members whose snapshots happened to be skewed.

A local group solicits new members by periodic polling. Members periodically send their requests to the next outer group. The group address, order number and error fingerprint of the inner group are carried in the polling request to solicit new members and helpers.

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 polling requests 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 the periodic polling requests scheduled by inner-group members suppress one another. If a requester does not receive a retransmission in its first try, the next repair request addressed to its outer group can also serve the purpose of membership solicitation.

Our mechanism requires that a member joins an outer local group before an inner local group. A helper does not solicit new comers into a local group since a helper does not maintain the order of nested local groups. If a member is solicited into an inner group without joining proper outer groups, it cannot received replies triggered by requests from outer groups. Consequently, its losses require multiple request iterations to recover, resulting in longer recovery delay and higher bandwidth overhead.

The same scheme is used to solicit new helpers. If a replier in the outer group responds to a polling request, it joins the corresponding inner group as a helper. However, if the request can be handled by a helper that is already in the inner group, this helper is closer to the requester and is most likely to respond first. Therefore, a new helper will rarely need to join the inner group unless all helpers in the inner group have left.

4 Simulation Results and Discussion

We believe the behavior of our proposed mechanisms can be best understood by first testing a variety of extreme settings before moving on to more general scenarios. In this section we first explore 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 five different session sizes: 8, 16, 32, 64 and 128. We simulated the performance of five different mechanisms for each session size: the global error recovery, the hop-scoped error recovery, and the group-scoped error recovery with three different degrees of loss sharing, 33%, 50% and 100%. We choose $\alpha = 0$ (error threshold to propose a local group), $k = 3$ (threshold for an inactive helper to leave a local group), $m = 10$ (number of losses in a loss-sharing measurement period) and $f = 6$ (number of losses in an error fingerprint) in all the group-scoped error recovery simulations. Each simulation starts with a warmup period and measures the error recovery activities of the next 2500 data losses. Data losses are generated by assigning an uniformly-distributed error rate on each lossy link of the simulated topologies, and these error rates are fixed throughout a single simulation. The aggregated error rate among all links is 10% of the traffic, including data, requests, replies and session messages. The data rate is 40 packets per second. The link delay is 10 ms for links between routers and 3 ms for links connecting group members. The session cycle time is 9 ± 1 seconds. The warmup period is defined as each lossy link experiencing at least 250 losses. The total simulation time is roughly 625 seconds.

The performance is evaluated by three metrics: the request traffic, the reply traffic and the recovery delay. The request traffic is the product of the average measured request scope and the average measured number of requests per loss. The request scope is a fraction of the global scope and it is measured in terms of the number of hops that a request propagates. For example, in the global error recovery, the request scope is equal to the global scope since each request is multicast to the entire session. The reply traffic is the product of the average measured reply scope and the average measured number of replies per loss. The recovery delay is measured in terms of the one-way propagation delay from the data source. In other words, it is the interval between a member's detection of a loss and reception of a retransmission, divided by the one-way propagation delay from the data source to the member.

In our simulations, we adopted a random timer adaptation mechanism to optimize the performance of the recovery delay, and the number of requests and replies per loss. The general idea is to make the generation of request and reply timers adaptive to the network and session environment. A member estimates the number of competing requesters and repliers by interpreting feedback from a session, and uses the estimated values to tune its request and reply timer parameters. These parameters determine whether requests and replies are generated aggressively or conservatively. Generally speaking, adopting local error recovery limits the number of competing requesters and repliers, which allows members to send requests and replies more aggressively and reduces recovery delays. More discussion is in [16, 17, 18].

4.1 Topologies with Multiple Lossy Links

The first set of simulations assumed that all links have uniformly-distributed error rates.

The Star Topology

Figure 6(a), 6(b) and 6(c) show the simulation results in the star topology. Members in the star topology have independent losses, hence there is no loss shared among members and approximately one request message per loss is generated. Since the distances between each pair of members are equal, the hop-scoped error recovery performs exactly like global error recovery. Note that the number of available helpers for a specific loss is large in the global error recovery and the hop-scoped error recovery, there are multiple replies generated per loss (Figure 6(b)).

In the group-scoped error recovery, each member creates its own local group. The requests and replies only propagate within individual local groups. Because of the constant number of helpers in a local group ($k = 3$), the request and reply traffic decreases by increasing session size.

The number of local groups in the 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 the group-scoped error recovery as $\frac{1}{n}$ of the traffic in the global error recovery. Since this estimated request and reply traffic is a lower bound, it represents the greatest degree of savings possible. [8] The estimated values are plotted as gray curves in Figure 6(a) and 6(b).

[8]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.

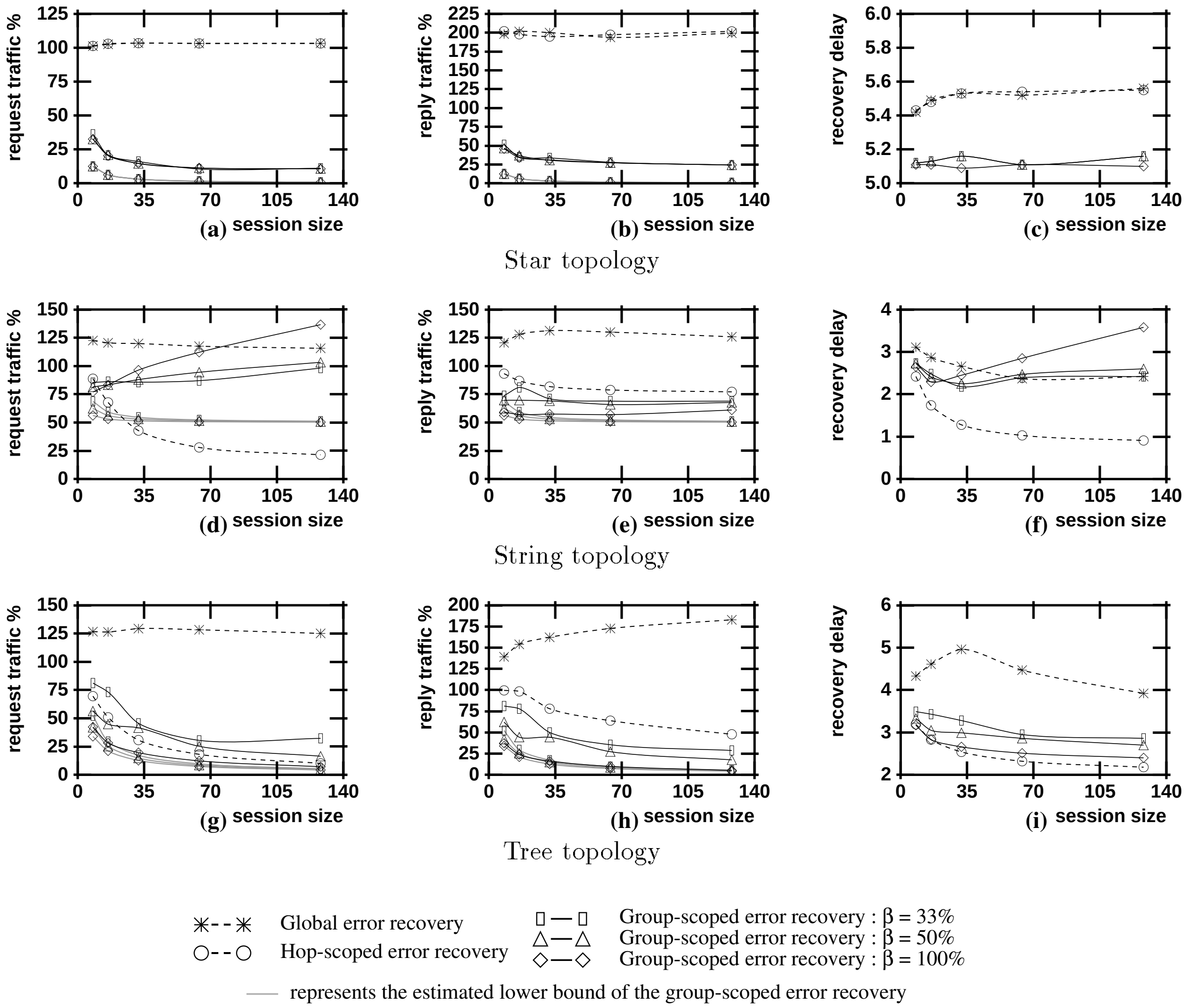

Figure 6: Simulation results of all links with uniformly-distributed error rates

The String Topology

In the string topology, downstream members share losses with all their upstream members. A downstream member can rely on its upstream members to ask for repair in the hop-scoped error recovery. As a result, request traffic is reduced significantly (Figure 6(d)).

Topology	Star topology					String topology					Tree topology				
Session size	8	16	32	64	128	8	16	32	64	128	8	16	32	64	128
Requests per loss	1.01	1.03	1.03	1.03	1.03	2.27	3.24	4.00	5.17	7.90	1.21	1.80	2.03	2.28	2.60
Request hops	16	32	64	128	256	6.26	6.71	6.86	6.92	6.95	9.17	9.06	9.68	10.18	9.99
Request traffic %	101.3	102.9	103.3	103.2	103.1	88.81	67.82	42.87	27.96	21.46	69.58	50.86	30.72	18.12	10.14

Table 1: The number of requests per loss and the number of hops that a request travels in the hop-scope error recovery

Table 1 shows the average number of requests per loss and the average number of hops that a request message travels in the hop-scoped error recovery. The number of requests per loss in the string topology increases with the session size, but 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, the average request traffic decreases with the session size. As described earlier, even if multiple requests per loss are presented in the 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.

Topology	Star topology					String topology					Tree topology				
Session size	8	16	32	64	128	8	16	32	64	128	8	16	32	64	128
Requests per loss	5.21	6.52	9.40	14.55	27.40	8.65	16.18	31.71	61.35	119.5	5.49	6.81	8.62	10.44	11.36
Request hops	1.00	1.00	1.00	1.00	1.00	1.42	1.65	1.95	2.34	2.93	1.23	1.32	1.46	1.51	1.62
Request traffic %	32.56	20.36	14.69	11.35	10.69	77.02	83.28	96.41	112.2	136.6	42.30	28.16	19.64	12.33	7.17

Table 2: The number of requests per loss and the number of hops that a request travels in the group-scope error recovery with $\beta = 100\%$

In the group-scoped error recovery, request messages propagate to all downstream members for suppression and a limited number of upstream neighbors for retransmission. A request may not reach any helpers if the scope of the request does not cover the lossy link where the packet was dropped. When the length of the string increases, the number of nested local recovery groups increases and it is more likely that requests from inner groups do not reach any helpers. Therefore, request traffic goes up with the session size as shown in Figure 6(d). Table 2 shows the average number of requests per loss and the average number of hops that a request message travels in the group-scoped error recovery with $\beta = 100\%$. The number of requests per loss and the average hops of a request in the string topology increase with the session size, thus the average request traffic increases with the session size.

In terms of reply traffic, since the hop-scoped error recovery does not regulate the direction in which the reply messages propagate, the hop-scoped error recovery should perform much worse than the group-scoped error recovery. However, in the group-scoped error recovery, a requester sends it second request to its outer group if the first one did not reach any helper. Since an outer-group member may already ask for repair, the second request from the inner-group member is very likely to trigger a duplicate reply. Consequently, the number of replies per loss increases and the improvement limited in comparison with the hop-scoped error recovery (Figure 6(e)).

The recovery delay in the hop-scoped error recovery decreases with the session size since request messages only reach a small number of members. As a consequence, members send requests more aggressively and the recovery delay is reduced (see the discussion of dynamic timer adaptation in Section 4). On the other hand, members have large estimated timer parameters in both the global error recovery and the group-scoped error recovery since requests reach all members who share the loss. However, in the group-scoped error recovery, only those members whose innermost local group covers the lossy link are eligible to request retransmission, therefore the average delay increases with the session size (Figure 6(f)).

The Tree Topology

The tree topology is a mixture of the star and string topologies. The average request traffic decreases with the session size in the group-scoped error recovery because the number of nested local groups is much smaller than it is in the string topology (Figure 6(g)). As shown in Table 2, both the number of requests per loss and the request hops increases much more slowly with the session size.

The number of local groups in the group-scoped error recovery in the string and tree topologies is proportional to β. For example, if $\beta = 100\%$, each local group is responsible for the error recovery of a single lossy links. The number of local groups is equal to the number of lossy links in the session. If $\beta = 50\%$, 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 $\beta \cdot n$, the number of lossy links covered by a local group is $\frac{1}{\beta}$, and the percentage of losses recovered by a local group is $\frac{1}{\beta \cdot n}$.

Therefore, the estimated error recovery traffic, T, can be calculated as $T = \frac{1}{\beta \cdot n} \cdot \sum_{i=1}^{\beta \cdot n} \frac{\delta(i)}{n}$, where $\delta(i)$ is the size of the i^{th} local group. For the string topology, $\delta(i) = n - \frac{i-1}{\beta}$. For the tree topology, $\delta(i) \simeq 2^{(\log_2 n - \log_2(\frac{i-1}{\beta}+1))}$. The estimated lower bounds of the error recovery traffic in string and tree topologies are,

$$T_{string} = \frac{\beta \cdot n + 1}{2 \cdot \beta \cdot n} \quad , \qquad T_{tree} = \sum_{i=1}^{\beta \cdot n} \frac{1}{(\beta + i - 1) \cdot n}$$

The estimated values are plotted as gray curves in Figure 6(d), 6(e), 6(g) and 6(h).

4.2 Topologies with Randomly-Selected Lossy Links

In the second set of simulations, randomly-selected $\frac{1}{8}$ of the links have uniformly-distributed error rates, which makes one lossy link in the 8-node topologies, two lossy links in the 16-node topologies, and so on. The simulation results are shown in Figure 7. Both the hop-scoped and group-scoped error recoveries outperform the global error recovery in terms of the request and reply traffic, except for the hop-scoped error recovery in the star topology.

Generally speaking, the hop-scoped error recovery performs better than the group-scoped error recovery in terms of the request traffic if members have an upstream-downstream relationship (*i.e.,* they share losses.) Downstream members can rely on the requests from their upstream members to ask for retransmission, therefore the hop-scoped error recovery generates less request traffic in the string and tree topologies than the group-scoped error recovery (Figure 7(d) and 7(g)).

On the other hand, since the hop-scoped error recovery does not regulate traffic direction, it does not perform well in terms of the reply traffic if the degree of connectivity in the topology is high. For example, in the star and tree topologies, the reply traffic generated by the hop-scoped error recovery populates in a much larger region than the reply traffic generated by the group-scoped error recovery (Figure 7(b) and 7(h)). Note that, in the 16-node string topology, the reply traffic in the hop-scoped error recovery is close to 100% because the randomly-selected lossy links are in the middle of the topology (Figure 7(e)).

Interestingly, the group-scoped error recovery with small β (*e.g.,* $\beta = 33\%$) not only generates more request and reply traffic, but also produces longer recovery delay, than the group-scoped error recovery with large β (*e.g.,* $\beta = 100\%$). For small β, a member's error fingerprint matches more easily with the error fingerprint of a local group with which it does not share losses. If the lossy links are sparsely distributed, a member who joins a non-loss-sharing group by accident is more likely to request retransmission from remote helpers. Therefore, small β produces more bandwidth consumption and longer recovery delay than large β (Figure 7(f) and 7(i)).

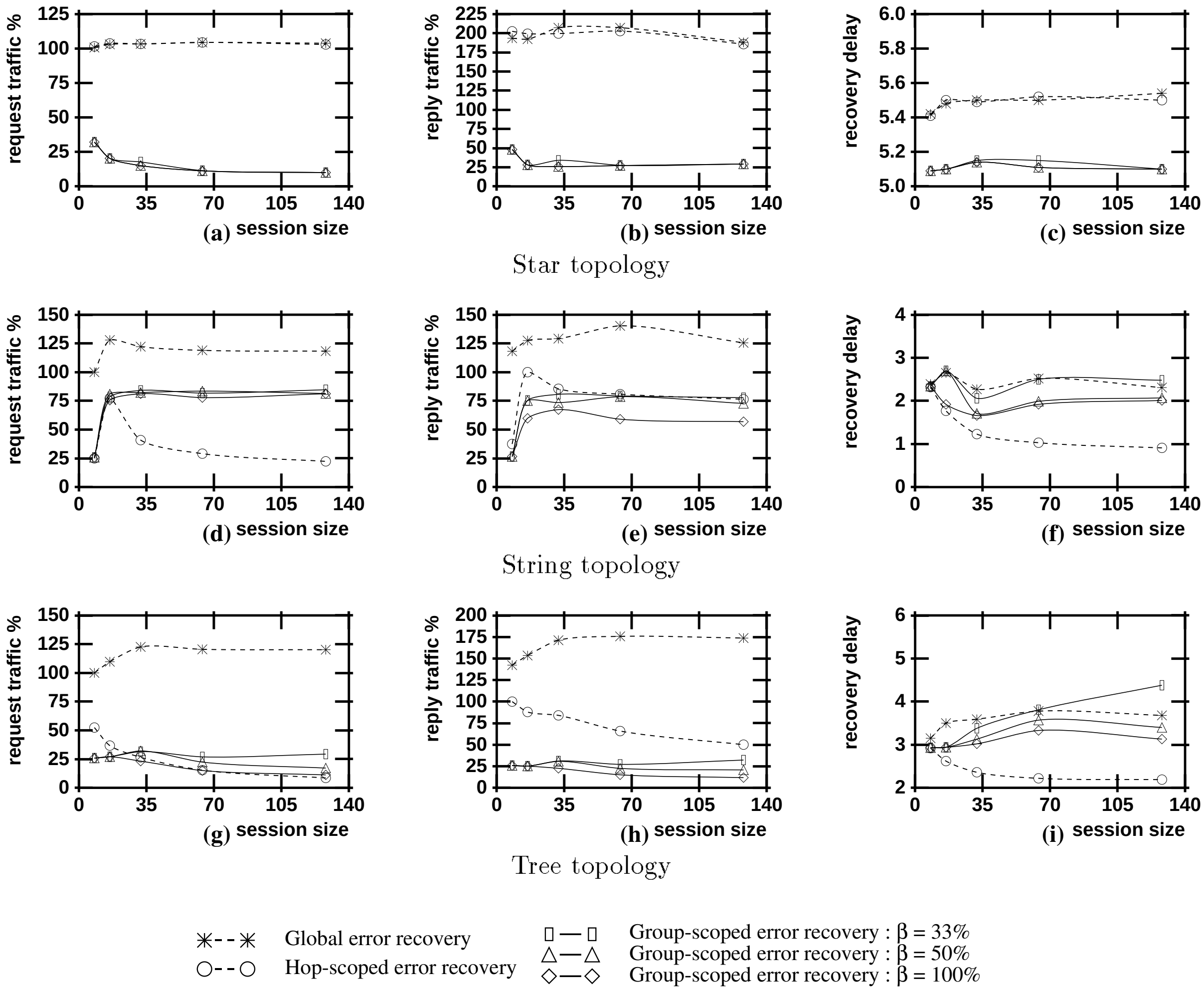

Figure 7: Simulation results of $\frac{1}{8}$ of the links with uniformly-distributed error rates

4.3 An Mbone-like Topology

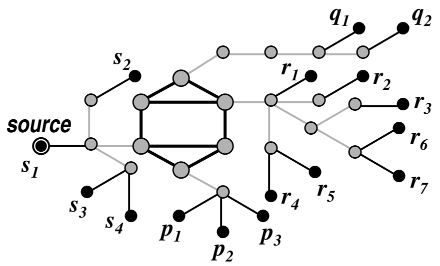

Figure 8: An Mbone-like topology (Lossy links are colored in gray and placed at local area networks.)

The local error recovery mechanisms were also simulated in an Mbone-like topology shown in Figure 8. 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, s_1, is selected as the data source. The lossy links are represented by gray lines. We assume most of the losses are at local area networks. The simulation results are shown in Table 3. In general, the group-scoped error recovery generates less request and reply traffic, and longer recovery delay than the hop-scoped error recovery.

Error recovery mechanisms	Global	Hop-scoped	Group-scoped β = 33%	Group-scoped β = 50%	Group-scoped β = 100%
Request scope %	113.14%	86.77%	55.92%	49.64%	39.24%
Reply scope %	139.75%	97.69%	59.24%	48.82%	40.08%
Recovery delay	5.15	4.21	4.47	4.80	4.80

Table 3: Simulation results of the Mbone-like topology

Figure 9 shows the average measured request and reply scopes of individual members. The scope is measured in terms of the number of hops that requests and replies travel. The hop-scoped error recovery performs in between the global error recovery and the group-scoped error recovery, however, the improvement in the reply scope is insignificant. The request and reply scopes in the group-scoped error recovery go down as β goes up. A large β means higher degree of loss sharing. As a consequence, fewer members are in each local group and less error recovery bandwidth is wasted. On the other hand, large β causes more local groups being created than small β. There are 10 local groups being used for error recovery in the sim-

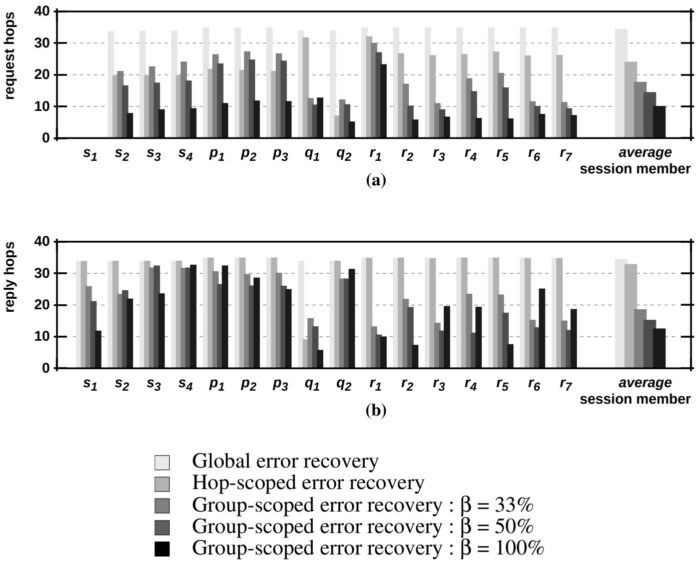

Figure 9: Average request and reply scopes of individual members in the Mbone-like topology

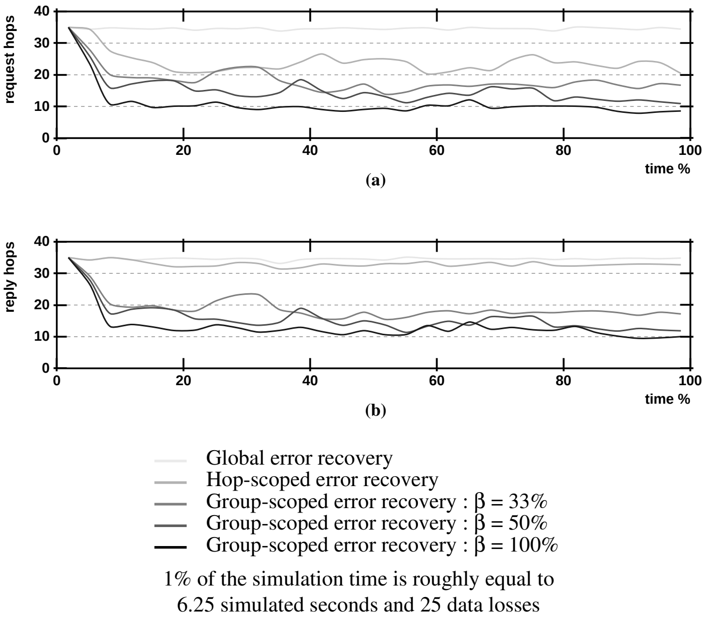

Figure 10: Error recovery traffic dynamics in the Mbone-like topology

ulation with $\beta = 100\%$, 6 local groups in the simulation with $\beta = 50\%$, and 3 local groups in the simulation with $\beta = 33\%$.

In Figure 9(a), p_1, p_2 and p_3 have relatively large request scopes when β is small (*e.g.,* 50% and 33%) because their local group includes remote members in other local area networks. q_1 and r_1 have relatively large request scopes in both the hop-scoped and the group-scoped error recoveries because their requests have to propagate across the Mbone to reach helpers. On the other hand, q_2 has relatively small request scopes because its requests only has to reach q_1. $r_2 \sim r_7$ have relatively large requests scopes because they are involved in the scenario sketched in Figure 2. They have to extend their request hop counts to suppress one another.

The reply scope, shown in Figure 9(b), depends on the origin of the request. A member with a small reply scope means most of the incoming requests are from its local area network; a member with a large reply scope means most of the incoming requests are across the Mbone. For example, $s_2 \sim s_4$, $p_1 \sim p_3$ and q_2 have relatively large reply scopes in both the hop-scoped and the group-scoped error recoveries, which means their replies respond to requests from remote members across the Mbone. On the other hand, the reply scopes of q_1 and r_1 are relatively small because they only responsible to recover losses of their downstream members within their local area network. Note that r_1's reply scope in the hop-scoped error recovery covers the entire topology because the hop-scoped error recovery does not regulate traffic direction. Response to remote requesters happens rarely in the group-scoped error recovery. As a result, the average reply scope is much smaller than it is in the global error recovery.

Figure 10 shows the measured request traffic and reply traffic during the simulation. It is measured in terms of the number of hops. The convergence periods of the request and reply scopes are relatively short in the group-scoped error recovery because once a member joined a local group, its request and reply traffic is reduced. The convergence time depends on the number of nested local groups and their sequence of creation. If the innermost group is created first, the convergence is fast. If the group with the largest scope is created first, this group has to be shrunk before the second nested group can be created (Only members without joining any local groups can propose the creation.) Therefore, it takes longer time to converge. In the hop-scoped error recovery, members require several session cycles to calculate the correct request and proxy hop counts. Generally speaking, the convergence time of request scope is approximately two session cycles. The computation of proxy scope relies on the results from other members, the convergence takes more than two session cycles.

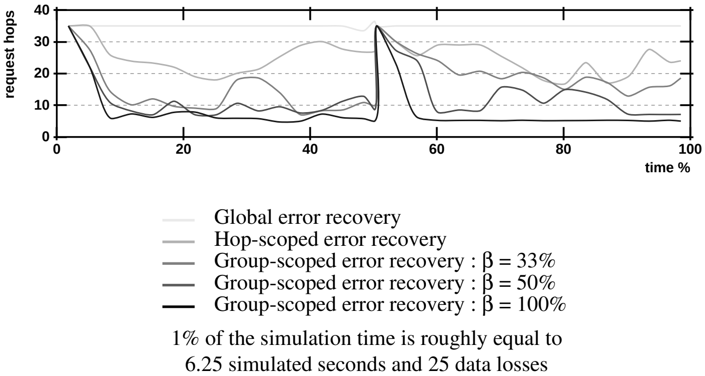

Figure 11: Request traffic dynamics of r_7 (r_7's state is reset in the middle of the simulation.)

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. r_7 starts with global error recovery after its state is reset. It calculates its request hop count based on the incoming session messages in the hop-scoped error recovery or learns of the existing local groups in the group-scoped error recovery to restore its state. The measured request traffic is shown in Figure 11.

4.4 A Random Topology with Sparsely-distributed Membership

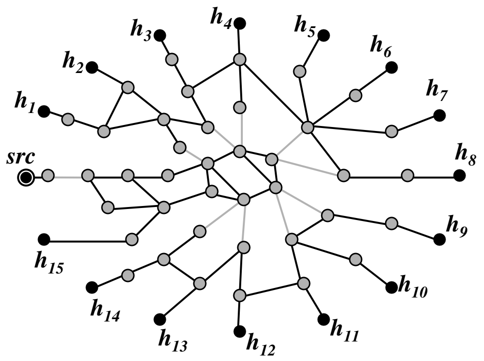

Figure 12: A topology with rich connectivity and sparsely-distributed membership (The lossy links are colored in gray are placed around the middle of the topology.)

Figure 12 shows a random topology used in our next simulation experiment. The topology consists of 40 routers, 52 inter-router links and 16 members. We place 9 lossy links around the middle of the topology and one lossy link close to the source, so members have shared losses as well as independent losses.

Error recovery mechanisms	Global	Hop-scoped	Group-scoped $\beta = 33\%$	Group-scoped $\beta = 50\%$	Group-scoped $\beta = 100\%$
Request scope %	105.86%	86.05%	40.90%	34.10%	25.00%
Reply scope %	128.36%	109.23%	49.40%	37.38%	23.88%
Recovery delay	6.33	3.63	3.56	3.58	3.35

Table 4: Simulation results of the random topology with sparsely-distributed membership

The simulation results are shown in Table 4. Since the degree of connectivity of routers are high, the request and reply delivery paths may differ from the data delivery path. In other words, members is more likely to find some close siblings to recover lost packets. Remember that we adopt a dynamic timer adjustment mechanism to optimize the error recovery timer. If requests and replies only propagate to near neighbors, the recovery timers are smaller than they are in the global error recovery. Therefore, the recovery delays in both the hop-scoped and group-scoped error recoveries are smaller than the recovery delay in the global error recovery. Note that the reduction of error recovery traffic is limited in the hop-scoped error recovery because its requests and replies travel more hops in the topology with rich connectivity and sparsely-distributed membership.

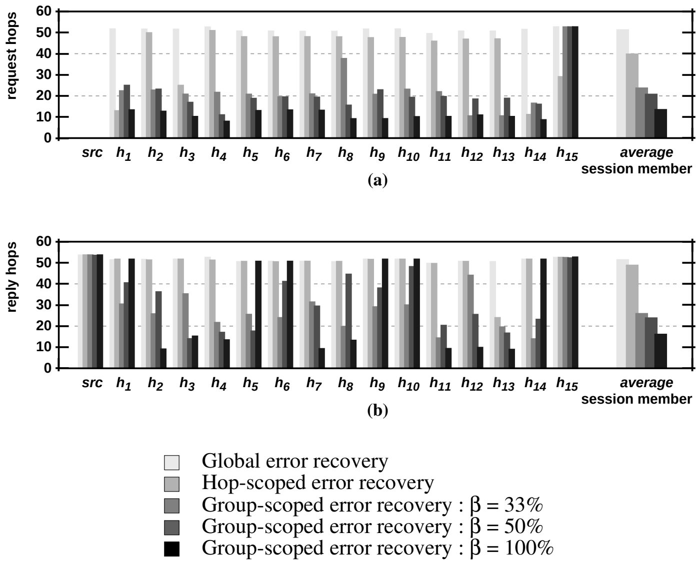

Figure 13: Average request and reply scopes of individual members in the random topology

Figure 13 shows the measured request and reply scopes of individual members. Note that member h_{15} has large request scope in the group-scoped error recovery because it shares losses with all other members. Its requests are populated in a local group which consists of all session members.

4.5 Discussion

In conclusion, we found that the group-scoped error recovery performs better than the hop-scoped error recovery in terms of the reply traffic. The hop-scoped error recovery performs better than the 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, and so the group-scoped error recovery appears to provide a better solution in terms of the traffic reduction. However, the number of duplicate requests increases with the number of nested groups; issues related to this scenario are left for future study.

If we consider other sources of overhead introduced by these two approaches, it appears that the group-scoped error recovery imposes more overhead on session members as well as the underlying multicast routing. For example, group-scoped error recovery requires the host to send periodic IGMP messages to refresh the multicast delivery path for each local group [19]. Issues related to the overhead of multicast group management are left for future study.

5 Related Work

There have been several other treatments of error recovery for reliable multicast transport [20, 21, 22]. In contrast to our proposal which assumes session members are autonomous, these previous works require various degree of static configuration, centralized coordination or router support.

Hofmann [23, 24, 25] 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 provides 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.

Towsley *et al.* [26] 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.

Holbrook *et al.* [27] 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.

TMTP [28] configures session members in the same subnet 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.

RMTP [29, 30] 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. 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 incoming requests.

PGM [31] makes use of the router support to maintain a tree hierarchy among routers. When a packet is lost, a request is unicast hop-by-hop upstream towards the source. Intermediate routers build retransmit states in order to remember where to forward the corresponding reply. Eventually, the request will hit the source or a member on the request forwarding path who has the requested packet, which triggers a reply multicast hop-by-hop downstream towards the requester.

Papadopoulos *et al.* [32] elaborated further on the router forwarding model to localized the scope of error recovery traffic. Each router selects a replier link which points towards a local replier. Instead of forwarding requests upstream towards the source, a router forwards a request towards its local replier if the incoming request is not received from the replier link. Since the replier link is a downstream link of the router, this router is at the turning point of the request delivery path. Eventually, the request reaches a replier and triggers a reply. The reply is first unicast to the turning point and then it is multicast downstream from the turning point.

6 Conclusion

We proposed two different approaches to reduce error recovery traffic in SRM. In the 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 the hop-scoped error recovery is relatively small. However, the hop-scoped error recovery does not regulate the direction of traffic propagation. If the topology of a session is star-shaped, the hop-scoped error recovery does not perform much better than the 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 recovery group, thus the error recovery traffic is only distributed within the local group. Group-scoped error recovery requires individual members to maintain and manage multiple local groups. Therefore, more overhead is imposed on members as well as on the underlying multicast routing.

There remain several open issues. In the 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 the group-scoped error recovery may not be acceptable. Further research should investigate group aggregation across sources. A local group is associated with one or more lossy links. Sources who share the delivery path (*e.g.,* shared-tree multicasting) and the lossy links along the path could be considered the same in terms of error recovery, and so error recovery from these sources should be handled by a single local group.

Another scenario that we have not fully understood is the convergence time of the group-scoped error recovery in the presence of network dynamics [33]. For example, if the network topology changes, members in a local group may not share the same lossy links, *i.e.,* they do not share losses anymore. 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. Members have to readjust themselves adaptive to these networks dynamics so that the new membership in the local group represents a set of members who share the same losses. The study of the convergence time of membership readjustment can help us to better understand the tolerance to network dynamics in our group-scoped scheme.

Finally, one might consider combining these two approaches by using hop-scoped request messages in local groups since the hop-scoped error recovery produces better request traffic reduction. However, requests in our hop-scoped scheme are addressed to the global session group with a specific hop count and that hop count is determined by measuring how far is the closest upstream neighbor. 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

[1] Sally Floyd, Van Jacobson, Ching-Gung Liu, Steve McCanne and Lixia Zhang. "A Reliable Multicast Framework for Lightweight Session and Application Layer Framing". *Proceeding of ACM SIGCOMM '95*. August 1995.

[2] Sally Floyd, Van Jacobson, Ching-Gung Liu, Steve McCanne and Lixia Zhang. "A Reliable Multicast Framework for Lightweight Session and Application Layer Framing". *IEEE/ACM Transactions on Networking, Volume 5, Number 6*. 1997.

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

[4] D. Clark, M. Lambert and L. Zhang. "NETBLT: A High Throughput Transport Protocol". *Proceedings of ACM SIGCOMM '87, Pages 353-359*. August 1987.

[5] Sridhar Pingali, Don Towsley and James Kurose. "A Comparison of sender-initiated and Receiver-Initiated Reliable Multicast Protocols. *Proceedings of ACM SIGMETRICS '94*. 1994.

[6] J. Nonnenmacher, M. Lacher, M. Jung, E. Biersack and G. Carle. "How bad is Reliable Multicast without Local Recovery?". *Proceeding of IEEE INFOCOM '98*. March 1998.

[7] Suchitra Raman, Steve McCanne and Scott Shenker. "Asymptotic Behavior of Global Recovery in SRM. *Proceedings of ACM SIGMETRICS '98/PERFORMANCE '98*. June, 1998.

[8] Puneet Sharma, Deborah Estrin, Sally Floyd and Van Jacobson. "Scalable Timers for Soft State Protocols". *Proceedings of the IEEE INFOCOM '97*. April, 1997.

[9] Puneet Sharma, Deborah Estrin, Sally Floyd and Lixia Zhang. "Scalable Session Messages in SRM Using Self-Configuration". *ftp://catarina.usc.edu/pub/puneetsh/papers/ssm.ps*. 1998.

[10] D. Waitzman, C. Partridge and S. Deering. "Distance Vector Multicast Routing Protocol". *RFC1075*. November 1988.

[11] S. Deering. "Multicast Routing in a Datagram Internetwork". *PhD Thesis, Stanford University, Palo Alto, CA*. December 1991.

[12] S. Deering, D. Estrin, D. Farinacci, V. Jacobson, C. Liu and L. Wei. "An Architecture for Wide-Area Multicast Routing". *Proceeding of ACM SIGCOMM '94*.

[13] S. Deering, D. Estrin, D. Farinacci, V. Jacobson, C. Liu and L. Wei. "The PIM Architecture for Wide-Area Multicast Routing". *IEEE/ACM Transactions on Networking, Volume 4, Number 2*. 1996.

[14] Dante DeLucia and Katia Obraczka. "Multicast Feedback Suppression Using Representatives". *Proceedings of IEEE INFOCOM '97*. 1997.

[15] Dante DeLucia and Katia Obraczka. "A Multicast Congestion Control Mechanism for Reliable Multicast". *Proceedings of IEEE ISCC '98*. 1998

[16] Ching-Gung Liu, Deborah Estrin, Scott Shenker and Lixia Zhang. "Timer Adjustment in SRM". *Technical report USC 97-656, University of Southern California*. July 1997.

[17] Ching-Gung Liu, Deborah Estrin, Scott Shenker and Lixia Zhang. "Recovery Timer Adaptation in SRM". *ftp://catarina.usc.edu/charley/papers/timer.ps*. 1997.

[18] Ching-Gung Liu. "Error Recovery in Scalable Reliable Multicast". *Ph.D. Dissertation, University of Southern California*. December 1997.

[19] S. Deering. "Host extensions for IP multicasting". *RFC1112*. August 1989.

[20] Brian Neil Levine and J.J. Garcia-Luna-Aceves. " A Comparison of Known Classes of Reliable Multicast Protocols". *Proceedings of International Conference on Network Protocols (ICNP-96)*. October 1996.

[21] A. Mankin, A. Romanow, S. Bradner and V. Paxson. "IETF Criteria for Evaluating Reliable Multicast Transport and Application Protocols". *RFC 2357*. 1998.

[22] Katia Obraczka. "Multicast Transport Mechanisms: A Survey and Taxonomy". *To appear in IEEE Communications Magazine*. 1998.

[23] M. Hofmann, T. Braun and C. Carle. "Multicast communication in large scale networks". *Proceedings of "Third IEEE Workshop on High Performance Communication Subsystems (HPCS)*, Mystic, Connecticut, August 1995

[24] Markus Hofmann. "A Generic Concept for Large Scale Multicast" *Proceedings of International Zurich Seminar on Digital Communication (IZS'96), Springer Verlag*. February 1996.

[25] Markus Hofmann. "Adding Scalability to Transport Level Multicast". *Proceedings of "Third COST 237 Workshop - Multimedia Telecommunications and Applications" (Springer Verlag), Barcelona, Spain*. November 1996.

[26] Sneha Kasera, Jim Kurose and Don Towsley. "Scalable Reliable Multicast Using Multiple Multicast Groups", *CMPSCI Technical Report TR 96-73*. October 1996.

[27] Hugh W. Holbrook, Sandeep K. Singhal and David R. Cheriton. "Log-Based Receiver-Reliable Multicast for Distributed Interactive Simulation". *Proceedings of ACM SIGCOMM '95*. August 1995.

[28] R. Yavatkar, J. Griffioen and M. Sudan. "A Reliable Dissemination Protocol for Interactive Collaborative Applications". *Proceedings of ACM Multimedia '95*. 1995.

[29] John C. Lin and Sanjoy Paul. "RMTP: A Reliable Multicast Transport Protocol". *Proceedings of IEEE INFOCOM '96, Pages 1414-1424*. April 1996.

[30] S. Paul, K. K. Sabnani, J. C. Lin and S. Bhattacharyya. "Reliable Multicast Transport Protocol (RMTP)". *To appear in IEEE Journal on Selected Areas in Communications, special issue on Network Support for Multipoint Communication.*

[31] Tony Speakman, Dino Farinacci, Steve Lin and Alex Tweedly. "PGM Reliable Transport Protocol Specification". *Internet Draft draft-speakman-pgm-spec-01.txt*. January, 1998

[32] C. Papadopoulos, G. Parulkar and G. Varghese. "An Error Scheme for Large-Scale Multicast Application". *Proceedings of IEEE INFOCOM '98*. March, 1998

[33] Kannan Varadhan, Deborah Estrin and Sally Floyd. "Impact of Network Dynamics on End-to-End Protocols: Case Studies in TCP and Reliable Multicast. *Technical report USC 98-672, University of Southern California*. 1998.

Ching-Gung Liu (M '98 / ACM '95) is currently a member of Research Staff of Fujitsu Laboratories of America, Inc., Sunnyvale, CA. He received his Ph.D. and M.S. from the University of Southern California in 1997 and 1991, and B.S. from National Taiwan University, Taiwan, in 1988. His research is focus on multicast routing, transport and application. He has been working on the design and implementation of PIM (Protocol Independent Multicast) and SRM for the past five years.

Deborah Estrin (S '78 - M '80 - SM '95) is a Professor of Computer

Science at the University of Southern California in Los Angeles where she joined the faculty in 1986. Estrin received her Ph.D. (1985) and M.S.(1982) from the Massachusetts Institute of Technology and her B.S. (1980) from U.C. Berkeley. In 1987, Estrin received the National Science Foundation, Presidential Young Investigator Award for her research in network interconnection and security. Estrin is a Co-PI on the DARPA Virtual Internet Testbed (VINT) project and the NSF Routing Arbiter project at USC's Information Sciences Institute where she spends much of her time supervising doctoral student research.

Estrin is an active member of the IETF multicast routing related working groups and a long-time member of the End-to-End research group. Estrin is a member of the ACM, IEEE,and AAAS. She has served on numerous panels for The National Science Foundation, The National Academy of Engineering, and served on DARPA's Information Systems and Technology (ISAT) advisory board from 1995-98. Estrin has served as an editor for the ACM/IEEE Transaction on Networks and the Wiley Journal of Internetworking Research and Experience, and as a member of the program committee for many IEEE INFOCOM and ACM SIGCOMM conferences, and she was program Co-Chair of ACM SIGCOMM '96.

Scott Shenker (S '87 - SM '95) is currently a Principal Scientist at the Xerox Palo Alto Research Center. He received his Sc. B (1978) from Brown University, his Ph. D. (1983) in theoretical physics from the University of Chicago, and spent the 1983-4 academic year at Cornell University as a Post-Doctoral Associate. His most recent computer science research focuses on the design of integrated services packet networks and the related issues of service models, scheduling algorithms, and reservation protocols. His recent economic research addresses incentive compatibility and fairness in various cost sharing mechanisms. Besides computer networks and theoretical economics, his other research interests include chaos in nonlinear systems, critical phenomena, distributed algorithms, conservative garbage collection, and performance analysis.

Lixia Zhang (S '81 - M '86 - SM '94) received the Ph.D. degree in Computer Sciences from the Massachusetts Institute of Technology, Cambridge, in 1989. She is an Associate Professor of Computer Science at the University of California, Los Angeles, where she joined the faculty in January of 1996. Prior to that, she was a member of Research Staff at Xerox PARC, engaged in research on advanced networking technologies, including high performance transport protocols, reliable multicast, and integrated services support over the Internet. She is the Co-Chair of the IETF RSVP Working Group, a member of the IETF Transport Area Directorate, and was a member of the Internet Architecture Board from 1994 to 1996.