

Recovery Timer Adaptation in SRM *

Ching-Gung Liu

Fujitsu Laboratories of America, Inc.
595, Lawrence Expressway, Sunnyvale, California 94086
Tel: (408) 567-4574 Fax: (408) 567-4558
charley@fla.fujitsu.com

Deborah Estrin

Computer Science Department / Information Science Institute
University of Southern California
Los Angeles, CA 90089
Tel: (310) 822-1511 ext. 253 Fax: (310) 823-6714
estrin@usc.edu

Scott Shenker

AT&T Center for Internet Research
International Computer Science Institute
1947, Center Street, Berkeley, California 84704
Tel: (510) 643-9153 Fax: (510) 643-7684
shenker@icsi.berkeley.edu

Lixia Zhang

Computer Science Department
University of California at Los Angeles
Los Angeles, CA 90095
Tel: (310) 825-2695 Fax: (310) 825-2273
lixia@cs.ucla.edu

Abstract

SRM is a generic framework for reliable multicast data delivery. The retransmission requests and replies used in SRM error recovery are multicast to the entire group. SRM uses random timers to avoid the message implosion problem, and the effectiveness of SRM's duplicate suppression depends critically on the design of these random timers. In order to achieve good performance in a dynamic network environment, the random timers should be adaptive to the network and session dynamics.

This paper focuses on the scaling behavior of the random timer adaptation mechanisms in SRM. Through analysis and simulations, we analyze the relationship between the timer setting parameters and performance, investigate the *representative timer adaptation approach* proposed in [1], and present a *collaborative approach* of random timer adjustment. Generally speaking, the representative approach elects representatives to send retransmission requests or replies; whereas the collaborative approach distributes the error recovery workload among members.

*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 Introduction

Scalable Reliable Multicast (SRM) [1] is one of the many proposed approaches to support reliable multicast data delivery. Scalability is defined as efficient error recovery, *i.e.*, low error recovery delay and few duplicate messages per loss, across the full range of group sizes and underlying network topologies. A key component of SRM is the use of random timers when sending retransmission requests and replies.

The original SRM timer algorithm (see [1]) incorporated both a deterministic and a probabilistic waiting period (*i.e.*, the timer was selected in an interval bounded away from zero). Due to the dynamic nature of the network and multicast groups, timer parameters should be adjusted dynamically.

In this paper, we focus on the scaling property of the random timer adaptation mechanisms in SRM. We start with the investigation of the relationship between the timer setting parameters and error recovery performance, and discuss the random timer adaptation mechanism proposed in the original SRM [1]. We refer to it as the *representative timer adaptation approach*. Using both analysis and simulation, we show that SRM can achieve near-optimal duplicate suppression if the probabilistic waiting period is proportional to the member’s neighborhood size, where neighborhood size refers to all the members who compete to send retransmission requests or replies regarding the same loss. These findings lead us to propose a new SRM timer scheduling scheme that produces near-constant recovery delay and duplicates per loss, regardless of the size of the multicast group. We call our mechanism the *collaborative approach* to random timer adjustment.

The paper is organized as follows. Section 2 gives a brief description of the SRM framework as described in [1], and the problems that we address in this paper. Section 3 looks into the relationship between the timer setting parameters and error recovery performance. Sections 4 discusses the representative approach to random timer adaptation in [1]. Section 5 describes the collaborative approach to random timer adaptation. Section 6 presents the simulation models and analyzes the simulation results, and Section 7 reviews related work. We conclude in Section 8 with a short summary.

2 Basic Approach of SRM

In this section, we give an overview of SRM as described in [1], emphasizing the features relevant to our work here. We use the term *session* to mean a multicast application that uses SRM as its underlying reliable multicast service. SRM provides only basic reliability support; it guarantees the eventual delivery of data to all members in the multicast session.

To avoid message implosion at the source in the error recovery process, SRM is receiver-initiated [3], with

each receiver 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.

When a member detects a packet loss, it waits for a random time period before sending its retransmission request. The random timer is scheduled in the time interval $A \cdot t_s \sim (A + B) \cdot t_s$, where t_s is the propagation delay between the requester and the data source, and A and B are the request timer parameters. When the timer expires, the scheduled request is multicast to the session group. If a request from another member is received or the currently scheduled request is sent, the requester exponentially backs off its request timer to ensure retransmission reliability. If a reply is received, the scheduled request is canceled. A member with the requested data ¹ responds to the request by scheduling a reply in the time interval $a \cdot t_p \sim (a + b) \cdot t_p$, where t_p is the propagation delay between the replier and the requester, and a and b are the reply timer parameters. When the timer expires, the scheduled reply is multicast to the session. The scheduled reply is canceled if a reply from another member is received while waiting.

3 Parameters and Performance

As pointed out in [1], there is not a single setting for the timer parameters that produces optimal performance across all network topologies and membership distributions. Furthermore, using constant timer parameters does not tune performance according to dynamic network topology and session membership. In this sections, we will investigate the relationship between timer setting parameters and error recovery performance. To simplify the description, our discussion will focus on the request parameters, A and B . The same analysis and conclusion can be applied to reply parameters, a and b , as well.

3.1 Duplicates Per Loss

3.1.1 Deterministic Suppression

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 [7]. The period of the SRM request timer consists of both a deterministic wait (*i.e.*, no messages are sent before $A \cdot t_s$, where t_s is the propagation delay from the source to the requester) and a probabilistic wait (the period between $A \cdot t_s$ and $(A + B) \cdot t_s$). Both

¹In addition to the original data source, SRM assumes, but does not mandate, other session members also save all the application data; those who do not save the data simply do not participate in the error recovery process.

portions of the wait are proportional to the propagation delay from a source. Thus, requesters far from a source have longer deterministic waiting periods than requesters near to the source. We call it *deterministic suppression* if a request sent at the maximal wait $((A+B) \cdot t_s)$ by a close-to-the-source member arrives before the deterministic waiting period of a farther-from-the-source member has expired; *i.e.*, scheduled requests (for messages from this particular source) at the distant requester will always be suppressed by requests from the nearer requester.

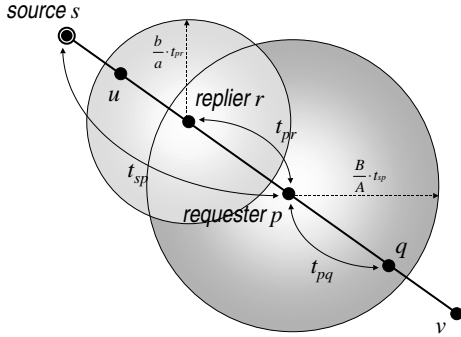


Figure 1: Radix defining the boundary of deterministic suppression and probabilistic suppression

Consider the scenario shown in Figure 1. A data packet sent by source s is lost between member r and member p . Member p detects the loss at time 0 and schedules a request before $(A+B) \cdot t_{sp}$ where t_{sp} is the propagation delay from s to p . Member q also detects the loss no later than time t_{pq} and schedules a request after $t_{pq} + A \cdot t_{sq}$. For p 's request to deterministically suppress q 's request, the latest time of p 's request arriving at q has to be sooner than q 's earliest request sending time; *i.e.*, $(A+B) \cdot t_{sp} + t_{pq} \leq t_{pq} + A \cdot t_{sq}$. We know $t_{sq} \leq t_{sp} + t_{pq}$ because q is farther from s than p , so we get $t_{pq} \geq \frac{B}{A} \cdot t_{sp}$.

The scheduled requests at member q will be suppressed by p 's request, deterministically, if q is farther from the source and its propagation delay from p is greater than $\frac{B}{A} \cdot t_{sp}$. If p 's requests arrive at q correctly, it is guaranteed that p 's requests will always suppress q 's requests.

Members who share the same losses and are within the radius of $\frac{B}{A} \cdot t_{sp}$ constitute the *requester neighborhood* of p with respect to source s because they compete with one another to send requests. There is no guarantee that p 's requests will always suppress others in its neighborhood. By applying the same analysis, the area without deterministic reply suppression is defined by a radius of $\frac{b}{a} \cdot t_{pr}$, for a member r with respect to a requester p . Members within the radius constitute the *replier neighborhood* of r with respect to source s .

3.1.2 Probabilistic Suppression

Deterministic suppression occurs only when the difference between the deterministic waiting periods of two

members is larger than the propagation delay between them. Thus deterministic suppression has no effect among members that are clustered together, in which case they have to rely on the randomization of their probabilistic waiting periods to suppress one another. We call this *probabilistic suppression*. Figure 2 illustrates this scenario.

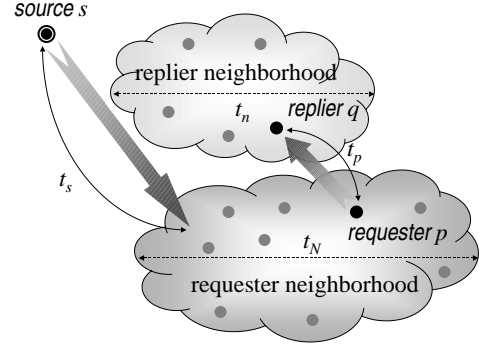


Figure 2: Members rely on probabilistic suppression within their requester and replier neighborhoods

In Figure 2, there are N members located within p 's requester neighborhood, with respect to losses from source s . Their propagation delays from s are all equal to t_s and the propagation delays between each pair of them are roughly t_N . Assuming a uniformly distributed random function is used to schedule requests, the probability that p 's request is not suppressed by any member within the neighborhood is $P = (\int_0^{t_N} 1 dx + \int_{t_N}^{B \cdot t_s} (\frac{B \cdot t_s - x + t_N}{B \cdot t_s})^{N-1} dx) / (B \cdot t_s) \leq \frac{t_N}{B \cdot t_s} + \frac{1}{N}$. Since there are N members within p 's requester neighborhood, the expected number of requests regarding a loss from source s is

$$E = P \cdot N \leq \frac{N}{B} \cdot \frac{t_N}{t_s} + 1 \quad (1)$$

Similarly, for a replier neighborhood with n members whose propagation delays from a requester p are t_p , and the propagation delays between each pair of them are roughly t_n , the expected number of replies regarding a request from p is $E' \leq \frac{n}{b} \cdot \frac{t_n}{t_p} + 1$.

3.1.3 Preventing Premature Requests

After sending a request, the requester backs off its request timer and schedules a second request to ensure retransmission reliability. The backoff request timer must be large enough to allow the reception of a reply. If the backoff timer expires before the reply to the first request has arrived, we say the backoff request is premature.

Consider the replier neighborhood in Figure 2. There are n repliers in the replier neighborhood and their propagation delays from the requester p are t_p . When a request from p arrives at the neighborhood, all repliers in the neighborhood will schedule replies between

$a \cdot t_p \sim (a + b) \cdot t_p$. The probability that the first expired reply times out after $(a + \lambda \cdot b) \cdot t_p$ is $(1 - \lambda)^n$, where $0 \leq \lambda \leq 1$. We want to compute the value of λ for which this probability is sufficiently small, for example, 5%. In other words, 95% of the first expired replies will time out before $(a + \lambda \cdot b) \cdot t_p$. We got $(1 - \lambda)^n \leq e^{-\lambda \cdot n} \leq 5\%$, therefore, $\lambda \simeq \frac{3}{n}$.

Since $0 \leq \lambda \leq 1$, the value of λ is bounded by $\min\{1, \frac{3}{n}\}$. It takes one-way delay to propagate the request to the replier and one-way delay to propagate the reply back to the requester. Hence, if the source is the replier in the worst case, the backoff request timer in p should be scheduled no sooner than $(2 + a + \lambda \cdot b) \cdot t_{sp}$, where t_{sp} is the propagation delay between the source and requester p , and $\lambda = \min\{1, \frac{3}{n}\}$. In other words, the parameter A for a backoff request timer should be at least equal to $(2 + a + \lambda \cdot b)$.

3.1.4 Ignoring Unsuppressed Requests

Request suppression and premature request prevention reduce, but does not eliminate, the chance of duplicate requests. When duplicate requests do occur, a mechanism to prevent response to unsuppressed requests is required.

A member responds to an unsuppressed request if it forgets a reply has already been sent for the same loss in response to a previous request. It is important for a member to remember the replies and ignore further requests regarding the same loss. A member should hold the records of its scheduled replies for a period of time and ignore unsuppressed duplicate requests. However, it should not ignore them forever since a reply may have been lost.

From the previous section, we identified that a backoff request for requester p is sent no sooner than $(2 + a + \lambda \cdot b) \cdot t_{sp}$. Therefore, the maximum hold time for a reply record triggered by p 's request should be no longer than $(2 + a + \lambda \cdot b) \cdot t_{sp}$, where t_{sp} is the propagation delay between the source and the original requester p and $\lambda = \min\{1, \frac{3}{n}\}$. During the hold period of a reply record, a member ignores further requests of the same loss.

3.2 Recovery Delay

Given A , B and the propagation delay from the source, if the number of requesters in a neighborhood competing to request retransmission is large, the average delay of the first expired request timer is shorter. For example, consider the case as shown in Figure 2 where there are N requesters in a requester neighborhood and their propagation delays from a source s are roughly t_s . Assuming a uniformly distributed random function is used to schedule requests, the expected waiting period of the first expired request timer within the neighborhood is

$$D = A \cdot t_s + \frac{N \cdot \int_0^{B \cdot t_s} x \cdot \left(\frac{B \cdot t_s - x}{B \cdot t_s}\right)^{N-1} dx}{B \cdot t_s} = A \cdot t_s + \frac{B}{N+1} \cdot t_s \quad (2)$$

The same analysis can be applied to the delay of the reply timers. If there are n members within a replier neighborhood and their delays from a requester p are roughly t_p , the expected waiting period of the first expired reply timer within the neighborhood is $D' = a \cdot t_p + \frac{b}{n+1} \cdot t_p$. The recovery delay is the sum of the request delay, the reply delay and the round-trip propagation delay between the requester and the replier. In the worst case where the source is the replier, an estimate of the recovery delay is given by $D + D' + 2 \cdot t_s$.

3.3 Summary

Since parameter A contributes the majority of request delay when there are many members, it is desirable to make A as small as possible. However, a small A increases the radius of the requester neighborhood and thus decreases the effectiveness of deterministic suppression. Fortunately, the number of duplicate requests can still be reduced by probabilistic suppression.

If the probabilistic waiting period is proportional to the member's neighborhood size, the number of duplicate requests and the recovery delay can both be properly controlled. The neighborhood size refers to the number of members who are competing to send retransmission requests or replies regarding the same loss. These findings lead us to propose a new SRM timer scheduling scheme which will be discussed in Section 5.

4 Representative Approach

[1] provide a simple mechanism for random timer adaptation. The adjustment algorithm of the request timer parameters, A and B , is shown in Table 1. The same algorithm is also used to adjust reply timer parameters.

```

A = B = 2
for each scheduled request in  $p$  for source  $s$ 
  if timer expires or is first reset
    then update avg_request_delay
  if a reply is received then update avg_dup_request
  if the request is sent then A- = 0.1
  if  $p$  is the closest requester then B- = 0.1
  elseif (avg_dup_request > dup_thresh)
    then A+ = 0.1; B+ = 0.5
  elseif (avg_dup_request < dup_thresh - 0.1)
    then if (avg_request_delay > delay_thresh)
      then B- = 0.1
      if (avg_dup_request < 0.25)
        then A- = 0.05
  else A+ = 0.05

```

Table 1: SRM dynamic adjustment algorithm for request timer parameter

In Table 1, $avg_dup_request$ is the average number of duplicate requests per loss, $avg_request_delay$ is the average waiting period of request timers. The average is computed as an exponential-weighted moving average with a weight equal to $\frac{1}{4}$. dup_thresh is the threshold for the number of duplicates per loss and it is equal to one. $delay_thresh$ is the threshold for the average request delay and it is equal to the round-trip propagation delay between a member and the source. The range to adjust A is $0.5 \sim 2$, *i.e.*, a member increases A at most to 2 and decreases A at most to 0.5. The range to adjust B is $1 \sim G$, where G is the session size

Generally speaking, the SRM adaptation mechanism differentiates the timer parameters among loss-sharing members. A member decreases its A Eventually, the timer parameters of a few members become sufficiently small to suppress requests from other members deterministically. These few members act like representatives and they dominate the activity of requesting retransmission. Interestingly, the SRM timer adaptation algorithm does not always produce representatives among members who share the same losses. Whether SRM exhibits this representative behavior depends on the initial neighborhood size.

As we have identified in Section 3.1.1, the requester neighborhood radius is proportional to the ratio of $\frac{B}{A}$. Therefore, the initial neighborhood size is determined by the initial values of A and B for a specific topology and membership distribution. If the initial neighborhood size is small, the member immediately behind the lossy link is more likely to be promoted as the representative since it has the advantage of sending its requests first. We refer to this as *leader mode* behavior. However, if the initial neighborhood size is large, a member's requests are less likely to expire first for the majority of the losses, even if it is closest to the lossy link. In this case, no member is promoted as the representative requester. To satisfy the duplicate threshold, members increase their timer parameters and rely on the randomization of their timers to minimize the number of duplicates. We call this behavior *group mode*, since no single member dominates in sending requests and replies.

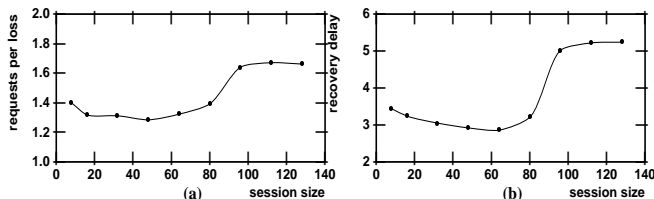


Figure 3: Initial neighborhood size determines the operating mode of the SRM timer adaption mechanism

Figure 3 shows simulation results for a binary tree topology. The data source is at the root of the tree topology and receivers are located at all tree nodes. One lossy link is selected near the source, hence, all members share the same losses throughout the simu-

lation. The propagation delay from the source to its nearest downstream member is ten times longer than the propagation delay between two neighboring members. As a result, the initial requester neighborhood includes all members in the session. The average recovery delay is measured in terms of the one-way propagation delay from the source.

Both the average requests per loss and the recovery delay plotted against session size are *S-shaped* curves. The lower edge of the curve represents leader mode where the nearest member to the source is promoted to be the representative. When the session size increases beyond 96, members operate in group mode, as illustrated by the upper edge of the curves.

5 Collaborative Approach

In Section 3.3, we found that when there are many members in the recovery neighborhood, a large value of A plays little role in suppressing the duplicates but increases the recovery delay. As we argue below, if B is selected properly the number of duplicate requests and the recovery delay can both be properly controlled even with a small value of A .

Consequently, we choose $A = a = 1$ ² and rely on probabilistic suppression to minimize the number of duplicates. The expected number of requests and the recovery delay in Equation 1 and 2 can be rewritten as, $E \propto \frac{N}{B} \cdot \frac{t_N}{t_s}$ and $D \propto \frac{B}{N} \cdot t_s$. Therefore, if we choose B as a linear function of the requester neighborhood size, *i.e.*, $B = C \cdot N$ where C is a constant, the expected number of duplicate requests is roughly proportional to $\frac{t_N}{t_s}$ and the request delay is constant in terms of the one-way propagation delay from the data source. Similarly, if we choose $b = c \cdot n$, where c is a constant and n is the size of the replier neighborhood, we get $E' \propto \frac{1}{c} \cdot \frac{t_n}{t_p}$ and $D' \propto c \cdot t_p$.

In this approach, members rely on probabilistic suppression to minimize the number of duplicates, and no member dominates the activities of sending requests and replies. Therefore, the error recovery workload should be fairly distributed among members. For this reason, we call it the collaborative approach of random timer adaptation.

5.1 Mechanism Description

Figure 4 shows the detailed control loop of our dynamic adjustment mechanism. Each member's feedback interpreter observes network feedback to estimate neighborhood size. We will discuss in more details in the following section. A parameter adjuster calculates the timer parameters from the estimated neighborhood size using the predefined linear functions (*i.e.*, $B = C \cdot N$

²The choice of A and a is an engineering decision based on simulation results.

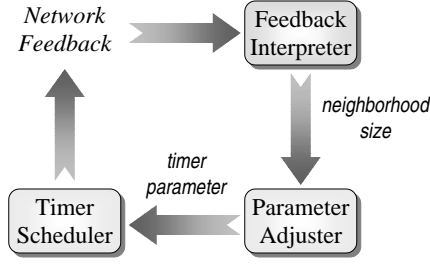


Figure 4: Control loop of dynamic timer adjustment

and $b = c \cdot n$); and a timer scheduler manages requests and replies with the new timer parameters.

In contrast with the representative approach in [1], which adjusts random timer primarily based on the average number of duplicates per loss, our collaborative approach uses linear functions to define the tradeoffs between the number of duplicates per loss and recovery delay. C and c are universally identical among members, however, C and c can be different from each other.

In Section 3.1.3 and 3.1.4, we identified that the minimum waiting period for the backoff request and the hold period of the reply record is equal to $(2 + a + \frac{3}{n} \cdot b) \cdot t_s$. Since $a = 1$ and $b = c \cdot n$, the waiting period for the backoff request and the hold period of the reply record in our timer adaption mechanism is equal to $(3 + 3 \cdot c) \cdot t_s$.

5.2 Neighborhood Size Interpretation

In this section, we will discuss the procedures of feedback interpretation. An interpreter can estimate the neighborhood sizes based on the recovery delay, or the number of requests and replies. We will discuss the interpretation from recovery delay only, the interpretation from the number of requests and replies is discussed in [8, 9]. In general, we find the approach of interpreting neighborhood size from duplicates is less accurate than interpreting neighborhood size from delay.

Since members have different neighborhood sizes, we will use the notation of N_p^s and B_p^s to refer to member p 's requester neighborhood size and request timer parameter with respect to the losses from source s , respectively. The notation of n_q^p and b_q^p are used to refer to member q 's replier neighborhood size, and reply parameter with respect to the requests from requester p , respectively.

From Equation 2, we know that the recovery delay of the first expired request timer in a requester neighborhood of N members is $(A + \frac{B}{N+1}) \cdot t_s$, on the average. If the first expired request timer has the value $(A + \vartheta \cdot B) \cdot t_s$ where $0 \leq \vartheta \leq 1$, then we can treat this as an estimate of N : $N = \frac{1}{\vartheta} - 1$. Thus, a member can interpret its requester neighborhood size from recovery delay if it identifies the first expired request and knows how long the first expired request has been scheduled. To solve this problem, members put the original value of their request timers in their outgoing requests, and a requester can simply collect these values in the incom-

ing requests and identify the smallest one as the first expired request.

Since we prefer a member near the source to send requests first, we want members near the source to produce smaller neighborhood size estimates than members far from the source. One simple way to achieve this is to weight ϑ by the delays from the source. For example, if both member p and r receive a request from q , the number of neighbors represented by q 's request (ϑ_q) is weighted by a factor of $\frac{t_{sp}}{t_{sq}}$ at member p , and it is weighted by a factor of $\frac{t_{sr}}{t_{sq}}$ at member r . As a result, p 's estimated neighborhood size is smaller than r 's neighborhood size by a factor of $\frac{t_{sp}}{t_{sr}}$. Note that, the estimated neighborhood size no longer represents the actual number of neighbors. Instead, it combines both the number of members competing to request retransmission in a requester neighborhood and their relative distance from the source.

The new requester neighborhood size is calculated by using an exponential-weighted moving average with a weight ω ³ between the previous neighborhood size and the new estimation. Note that, since the feedback from the network is ϑ , a member should perform the exponential-weighted moving average on ϑ before converting ϑ to the estimated neighborhood size, N . If ϑ is converted to N before the exponential-weighted moving average is performed, the result is divergent.

```

for each scheduled request in  $p$  for source  $s$ 
   $\vartheta = 1$ 
  for each request received from  $q$  (including  $p$  itself)
     $\vartheta = \min\{\vartheta, \vartheta_q \cdot \frac{t_{sq}}{t_{sp}}\}$ 
   $\Theta_p^s = (1 - \omega) \cdot \Theta_p^s + \omega \cdot \vartheta$ 
   $N_p^s = \frac{1}{\Theta_p^s} - 1$ 

```

Table 2: Algorithm of interpreting requester neighborhood size from recovery delay

The algorithm shown in Table 2 adapts the requester neighborhood size to the dynamic changes in a multicast session. The requester neighborhood size is adjusted for individual sources, and it is measured for individual losses. A measurement period starts when a loss is detected and it ends when a reply is received. Note that, the requester neighborhood size should be adjusted on a per-lossy-link, per-source basis. It is an approximation to adjust requester neighborhood size on a per-source basis only. If there are multiple lossy links between source s and member p , N_p^s is the average of the neighborhood sizes of all the upstream lossy links.

We can apply the same mechanism to estimate the size of a replier neighborhood. The measurement period of the replier neighborhood size is equal to the hold time of a scheduled reply, discussed in Section 3.1.4. It starts when a request is received and it ends when the scheduled reply is cleared. Note that, if a replier receives multiple requests for the same loss, the estimation is

³We choose $\omega = \frac{1}{8}$ in our simulations.

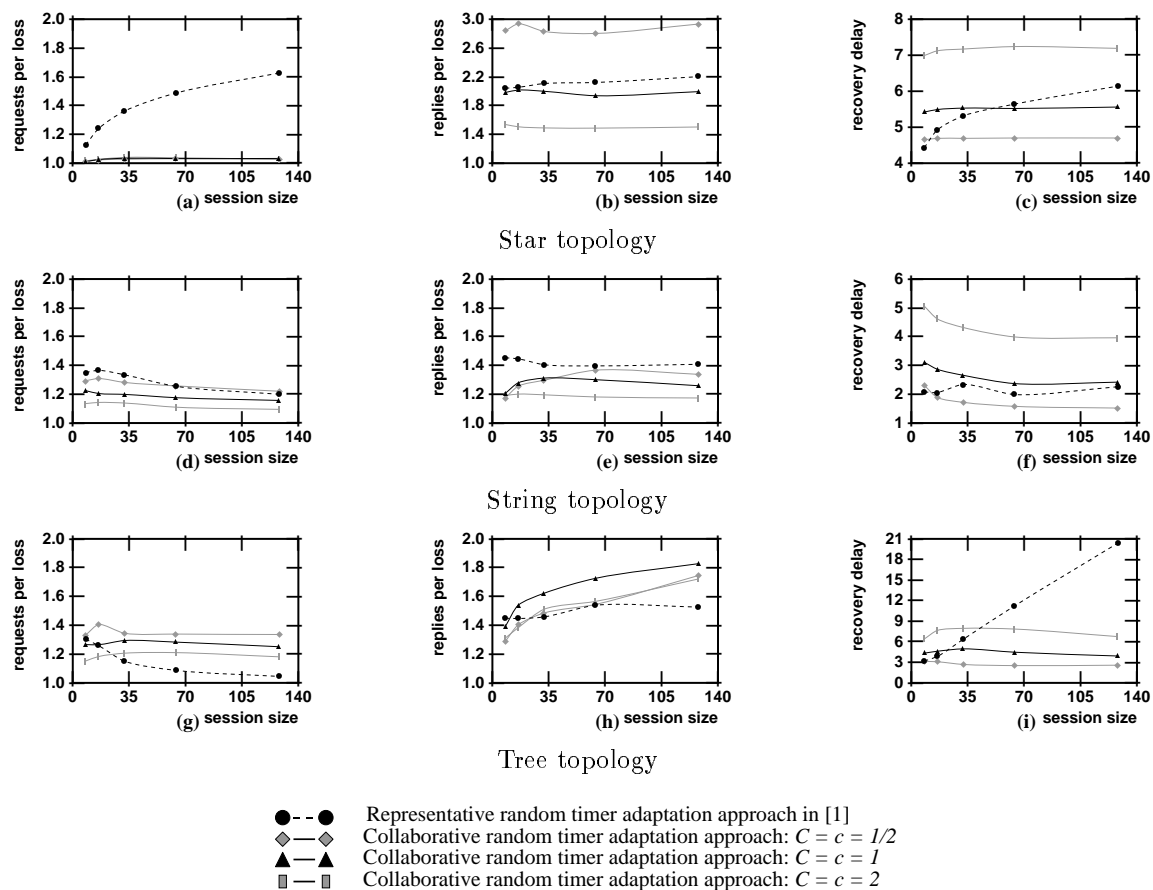


Figure 5: Simulation results of all links with uniformly-distributed error rates

for the requester whose request is received first.

6 Simulation Results

We initially explored our revised timer scheduling scheme and dynamic timer parameter adjustment 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 equal distance to the source and are siblings of one another. The string topology represents a session where all members have upstream/downstream relationship and share the data delivery paths with their upstream members. The binary tree topology represents a mixture of both.

We simulate the performance of the SRM adaptation mechanism suggested in [1] and our collaborative version of the timer adaptation mechanism. To simplify the description in the following sections, we will call them RA_{SRM} and CA_{SRM} , respectively.

Note that, the recovery delay is measured in terms of the one-way propagation delay from the data source; more precisely, the recovery delay 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.

6.1 Topologies with All Lossy Links

We simulate both timer adaptation approaches on the same topologies with multiple lossy links. Other loss patterns are discussed in [10]. Each link has a uniformly-distributed error rate and its error rate is fixed throughout the simulation.

6.1.1 Star Topology

The average requests per loss is close to one in CA_{SRM} . However, since the replies are operated in group mode in RA_{SRM} , the average waiting period of the reply timer is long. As a consequence, the delay of the backoff request timers are not sufficient to prevent premature requests and the average number of requests increases with the session size (Figure 5a). Furthermore, more duplicate requests produce larger request parameters. Thus, the recovery delay also increases with the session size (Figure 5c).

6.1.2 String Topology

In CA_{SRM} , even though members do not share identical losses and their estimated neighborhood sizes are the average of the actual neighborhood sizes associated with individual upstream lossy links, the average requests and replies per loss are still near-constant and

the recovery delay decreases with the session size.

Since each member has a portion of its losses that are not shared with others, RA_{SRM} is less effective in the all-lossy-link case because a member's independent losses will reduce its request timer parameters and the difference between the representative's parameters and others is less significant. Members far from the lossy link may send requests and replies, thus, the recovery delay does not decrease with the session size.

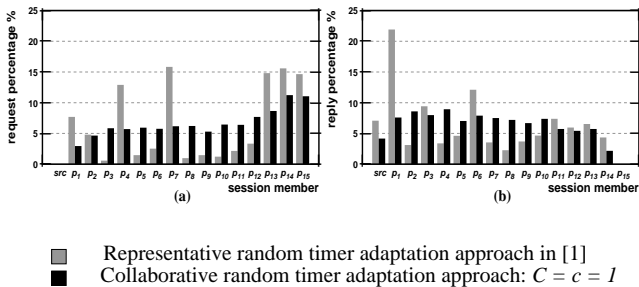


Figure 6: Distributions of the requests and replies among members in 16-node string topology

Figure 6 shows the request and reply distribution in the 16-node string topology. Unlike the single-lossy-link case, several representative requesters and repliers are selected in RA_{SRM} . Note that, most requests and replies are distributed towards the edges of the string topology.

6.1.3 Tree Topology

In CA_{SRM} , we notice that the number of requests decreases with the session size and the number of replies increases with the session size. Since the width increases exponentially with depth in the tree topology, more losses are distributed towards the leaf members when the session size increases. Therefore, fewer losses are shared by a majority of the members and the number of requests per loss decreases. On the other hand, since more losses are distributed towards the leaf members, more requests can be replied by a majority of the members. Note that, the average replies per loss when $C = c = 1$ is larger than the average replies when $C = c = \frac{1}{2}$. We identified this scenario as *local minima* in [9].

For the same reason mentioned above, the average requests per loss decreases with the session size and the average replies per loss increases with the session size in RA_{SRM} . However, the recovery delay increases substantially with the session size. To further understand this behavior, we plot the request and reply distribution of the 16-node tree topology in Figure 7.

From Figure 7, we found most of the requests are sent by leaf members (member $p_8 \sim p_{15}$) and none of the requests are sent by the intermediate members (member $p_2 \sim p_7$). Furthermore, almost all the replies are sent by member p_2 and p_3 . In other words, the majority of the reply timers are based on propagation delay that

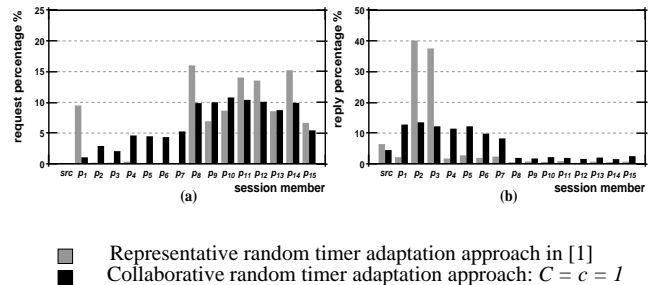


Figure 7: Distributions of the requests and replies among members in 16-node tree topology

is proportional to the depth of the tree topology. As a result, the recovery delay is sub-optimal.

Note that, RA_{SRM} operates in leader mode between the leaf members and others. However, it operates in group mode among leaf members. That is, the leaf members (*i.e.*, the representatives) rely on probabilistic suppression to reduce duplicates. Therefore, the leaf members' A 's are close to 0.5 and their B 's are proportional to the neighborhood size.

6.2 Individual Performance Analysis

The previous sections discussed the aggregated error recovery performance of session members. In this section, we will investigate member's individual behaviors. We choose the 8-node tree topology to examine the performance. The propagation delay from the source to its nearest downstream member is ten times longer than the propagation delay between two neighboring members. Therefore, the initial requester neighborhood includes all members in the session. We assume the session size is three times of the neighborhood size, *i.e.*, $G = 24$.

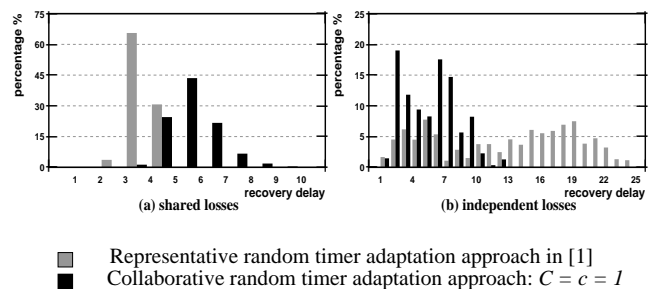


Figure 8: Recovery delay distribution of the shared losses and independent losses at a leaf member in 8-node tree topology

In the first simulation, each member has 1% of its losses that are not shared with others. We plot the request distributions of the shared losses and independent losses at a leaf member in Figure 8. For shared losses, the leaf member relies on the requests from the representative to ask for repair in RA_{SRM} . The request distribution of the shared losses is more tightly bounded than the distribution in CA_{SRM} .

The request delay for the leaf member’s independent losses are based on its own request timers. In RA_{SRM} , since the leaf member is not the representative and its parameters increase substantially during the process of the representative election, the request distribution of its independent losses spreads over a wide range. On the other hand, members’ timer parameters are similar to one another in CA_{SRM} . The request distribution has a tighter bound than the distribution in RA_{SRM} .

In the next simulation, we test the response of the two approaches to the changes of lossy links. All members share identical losses at the beginning and then we change the lossy link to the incoming link of one of the leaf members. In other words, the leaf member shares losses with others during the first half of the simulation and its losses are totally independent of others during the second half of the simulation.

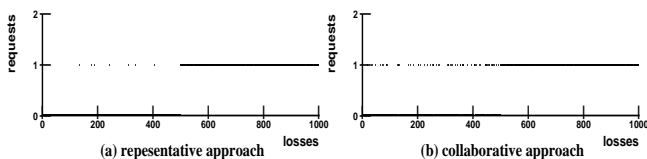


Figure 9: Number of requests for each loss sent by the leaf member

Figure 9 shows the distribution of the requests sent by the leaf member. The leaf member sends very few requests during the first half of the simulation in RA_{SRM} since most of the requests are sent by the representative (*i.e.*, the member closest to the source).

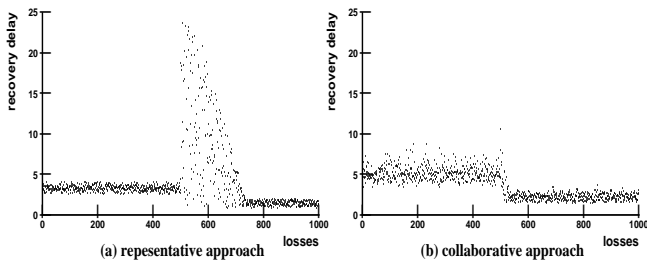


Figure 10: Recovery delay for each loss of the leaf member

Figure 10 shows the distribution of the recovery delay for each loss. During the first half of the simulation, the recovery delay is more tightly bounded in RA_{SRM} than it is in CA_{SRM} . However, The recovery delay is widely spread after the lossy link shifts in RA_{SRM} . On the other hand, CA_{SRM} is very sensitive in terms of the awareness of lossy link changes, while the transition period is almost unnoticeable.

6.3 Discussion

In conclusion, the representative approach performs extremely well if the number of lossy links is relatively small in comparison with the number of loss-sharing members. However, when the number of lossy links is

large, with the current parameter setting the representative approach becomes less likely to promote members in the optimal location to behave as representatives.

We believe the aggressiveness comes from the adaptation speed (*e.g.*, the amount of adjustment in each measurement). Members near the source are more likely to become representatives if the adaptation speed is slow. However, slower adaptation would make adjustment to topology changes even slower. The issues related to the tradeoffs of the adaptation speed in the representative approach clearly deserve further study.

The collaborative approach presents a well-behaved scaling property in all cases, however its performance is not as optimal as the representative approach in some cases.

7 Related Work

Most of the error recovery mechanisms in the proposed reliable multicast protocols focus on the avoidance of message implosion. Generally speaking, they can be categorized into structure-based and timer-based approaches [11]. In the structure-based approach, a subset of members are selected either to organize the error recovery activities or to process error recovery messages. In the timer-based approach, all members run the error recovery algorithm and they rely on the randomization of timers to suppress duplicate error recovery messages. A few examples are discussed below.

RBP [12] is a token-based reliable multicast protocol. A token circulates among members in a round-robin fashion to distribute the workload. The member possessing the token becomes the current token site. RBP adopts a sender-based error control mechanism between senders and token sites, and a receiver-initiated error recovery mechanism between token sites and receivers. The current token site is responsible for acknowledging each data reception. It is also responsible for recovering losses for other members and replies to the retransmission requests.

In MTP [13], time is divided into slots, called heartbeats, for data transmission. A fixed number of messages are sent in each time slot, including both the new data and the replies. A master is elected among all sources for the purpose of granting tokens. A member must obtain the token to become the sender of the current slot. At any point in time, only one data source can send data. The retransmission requests are unicast to sources and the replies are multicast to the whole group.

Holbrook *et al.* [14] suggested a hierarchic logging server structure to distribute the error recovery workload. Logging servers acknowledge each data reception from the source and they are responsible for recovering losses for other receivers. 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 reply based on the number of requests it receives.

In RMTP [15, 16], data are explicitly acknowledged by the receivers. To avoid ACK implosion, members are grouped into local regions and local regions are constructed into a tree hierarchy. A designated receiver (DR) is selected in each region, it is responsible for processing ACKs from its local region and acknowledging the data reception to its parent DR. DRs cache received data and respond to retransmission requests in their local regions. A reply is either unicast or multicast based on the number of requests per loss.

TMTP [17] has a similar flavor to RMTP in terms of the hierarchic error recovery structure. It groups members into domains and organizes domains into a hierarchic control tree. 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 limiting the TTL.

Grossglauser presents DTRM [18] to compute deterministic timer values based on the multicast tree topology and source-to-receiver propagation delays. For a single loss, the deterministic timers ensure that one member sends a request fast enough so the reply triggered by the request will arrive at other members before their request timers expire.

8 Conclusion

We investigated the relationship between the timer setting parameters and error recovery performance in SRM, discussed the representative timer adaptation approach proposed in [1], and presented a collaborative approach of random timer adjustment. Generally speaking, the representative approach elected representative to send retransmission requests or replies; whereas the collaborative approach distributes the error recovery workload among members.

Both analysis and simulations shows that the collaborative approach possesses well-behaved scaling property. In fact, by computing the timer parameters using linear functions from the neighborhood sizes, the number of requests (and replies) per loss and the recovery delay are not significantly affected by the session size. On the other hand, the operating mode of the timer adaptation mechanism in [1] is determined by the initial neighborhood size. It produces optimal performance in leader mode if the representative is near the lossy link, and acceptable performance in group mode. However, if a remote member is promoted to be the representative, both the average recovery delay and the scaling property are sub-optimal.

We believe the hypothesis of adaptive adjustment is to assume that past experience is a good prediction of the future. If there are multiple lossy links along a path,

members face different neighborhoods for losses at different lossy links. Therefore, the adaptive adjustment mechanism should be less aggressive so the estimation from the past can predict the average behavior of the future. From the simulation results, the representative approach is less robust than the collaborative approach, in terms of scalability, while the collaborative approach can be less efficient under small number of lossy links.

However, the behavior of both approaches are not yet fully understood. More extensive research effort is required.

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". *IEEE/ACM Transactions on Networking*. 1997.
- [2] D. Clark and D.Tennenhouse. "Architectural Considerations for a New Generation of Protocols". *Proceedings of ACM SIGCOMM '90, Pages 201-208*. September 1990.
- [3] Sridhar Pingali, Don Towsley and James Kurose. "A Comparison of sender-initiated and Receiver-Initiated Reliable Multicast Protocols". *Proceedings of ACM SIGMETRICS '94, Pages 221-230*. 1994.
- [4] Puneet Sharma, Deborah Estrin, Sally Floyd and Van Jacobson. "Scalable Timers for Soft State Protocols". *Proceedings of the IEEE INFOCOM '97*. April, 1997.
- [5] Puneet Sharma, Deborah Estrin, Sally Floyd and Lixia Zhang. "Scalable Session Messages in SRM". <ftp://catarina.usc.edu/pub/puneetsh/papers/infocom98.ps>. 1998.
- [6] D. Clark, M. Lambert and L. Zhang. "NETBLT: A High Throughput Transport Protocol". *Proceedings of ACM SIGCOMM '87, Pages 353-359*. August 1987.
- [7] S. Deering. "Host extensions for IP multicasting". *Internet Draft, RFC1112*. August 1989.
- [8] Ching-Gung Liu. "Error Recovery in Scalable Reliable Multicast". *Ph.D. Dissertation, University of Southern California*. December 1997.
- [9] Ching-Gung Liu, Deborah Estrin, Scott Shenker and Lixia Zhang. "Timer Adjustment in SRM". *Technical report USC 97-656, University of Southern California*. August 1997.
- [10] Ching-Gung Liu, Deborah Estrin, Scott Shenker and Lixia Zhang. "Recovery Timer Adaptation in SRM". *Technical report USC 97-664, University of Southern California*. August 1997.
- [11] 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.
- [12] J. Chang and N. F. Maxemchuk. "Reliable Broadcast Protocols". *IEEE/ACM Transactions on Computer Systems, Vol.2, No. 3, pp. 251-275*. August 1984.

- [13] S. Armstrong, A. Freier and K. Marzullo. "Multicast Transport Protocol". *Internet Draft, RFC1301*. February 1992.
- [14] 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.
- [15] John C. Lin and Sanjoy Paul. "RMTP: A Reliable Multicast Transport Protocol". *Proceedings of IEEE INFOCOM '96, Pages 1414-1424*. April 1996.
- [16] 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*.
- [17] R. Yavatkar, J. Griffioen and M. Sudan. "A Reliable Dissemination Protocol for Interactive Collaborative Applications". *Proceedings of ACM Multimedia 95*. 1995.
- [18] M. Grossglauser. "Optimal Deterministic Timeouts for Reliable Scalable Multicast". *Proceedings of IEEE INFOCOM 1996, pp. 1425-1432*. April 1996.