Timer Adjustment in SRM [*]

Ching-Gung Liu (USC),
Deborah Estrin (USC/ISI),
Scott Shenker (Xerox),
Lixia Zhang (UCLA/Xerox)

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.

This paper investigates, through analysis and simulation, the relationship between the timer setting parameters and error recovery performance in SRM. The performance metrics are error recovery delay and duplicates per loss. We propose an algorithm where the waiting period is proportional to a member's neighborhood size, and where a member's estimate of neighborhood size based on the observations of current performance. We find that this algorithm achieves near-optimal duplicate suppression. Moreover, as the size of the global group grows the recovery delay and duplicates per loss remain bounded.

1 Introduction

Scalable Reliable Multicast (SRM) [1] is one of the many proposed approaches to support reliable multicast data delivery. The main focus of SRM is to achieve scalability defined as efficient error recovery (low error recovery delay and few duplicate messages per loss) across full range of group sizes and underlying network topologies. As we explain later in Section 2 of this paper, a key component of SRM is the use of random timers when sending retransmission requests and replies. In this paper, we investigate the relationship between the timer setting parameters and error recovery performance.

The original (see [1]) SRM timer algorithm incorporated both a deterministic and a probabilistic waiting period (*i.e.*, the timer was selected in an interval bounded away from zero). Using both analysis and simulation, we find that removing the deterministic wait in timer setting from the random timer reduces recovery delay

[*]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.

significantly without degrading duplicate suppression. Moreover, we find that near-optimal duplicate suppression can be achieved if the probabilistic waiting period is proportional to the member's neighborhood size. 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 that produces a near-constant recovery delay [1] and duplicates per loss, regardless how large the size of the multicast group.

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. Section 4 proposes a revised timer scheduling scheme. Section 5 introduces our dynamic mechanisms to measure the neighborhood size and adjust the timer setting parameters. Section 6 presents the simulation models and analyzes the simulation results, and Section 7 reviews related works. 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. More stringent reliability functionalities, such as total ordering and fate sharing [2], if desired, are left to the application itself [3].

To avoid message implosion at the source in the error recovery process, SRM is receiver-initiated [4] with each receiver being responsible for detecting data losses and requesting retransmissions. SRM also adopts the approach of "multicasting everything" to maximize the collaboration among members in the process of error recovery. Requests and replies are multicast to all members in the session. Multicasting a request allows the nearest member with the requested data to send a reply first; it also suppresses other members from sending duplicate requests for the same data. Similarly, multicasting a reply suppresses duplicate replies and also delivers the reply to all members who suffer the loss without requiring the replier to know their individual identities or locations.

The SRM mechanisms can be decomposed into two parts: group state synchronization and receiver-initiated error recovery. Members periodically exchange session messages to report current group state (e.g., the highest received sequence number from each source) and to determine the propagation delays between each pair of members [3]. Members use group state information to detect data losses. This

[1]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.

[2]Fate sharing is when a multicast session terminates if a single member, or a specific subset of members in the session fail, depending on the semantics of the application.

[3]There is a semantic difference between the delay from a member and the delay to a member. We

is critical for the receiver-initiated error recovery approach because members do not otherwise know what has been sent to the session group. Each member uses the propagation delay information when scheduling its request or reply timers.

When a packet is lost, each member detecting the loss waits a random time period before sending its retransmission request. The random timer is scheduled in the time interval $A \cdot T \sim (A+B) \cdot T$, where T is the propagation delay between the requester and the data source, and A and B are constants [4]. When the timer expires, the scheduled request is multicast to the session group. If a duplicate request 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 [5] responds to the request by scheduling a reply in the time interval $a \cdot t \sim (a+b) \cdot t$, where t is the propagation delay between the replier and the requester, and a and b are constants. When the timer expires, the scheduled reply is multicast to the session. The scheduled reply is canceled if a duplicate reply is received while waiting.

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 [5]. The period of the SRM request timer consists of both a deterministic wait (*i.e.,* no messages are sent before $A \cdot T$) and a probabilistic wait (the period between $A \cdot T$ and $(A+B) \cdot T$). Both 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 ($(A+B) \cdot T$) by a close-to-the-source member arrives before the deterministic wait of a farther-from-the-source member has expired; scheduled requests (for messages from this particular source) at the distant requester will always be suppressed by requests from the nearer requester.

Deterministic suppression can only occur if the difference between the deterministic waiting periods of two members is large relative to the propagation delay between them, and thus deterministic suppression is comparatively rare. Usually, members have to rely on the randomization of their probabilistic waiting periods to suppress one another. We call this probabilistic suppression.

The same concepts of deterministic and probabilistic suppression can be applied to the reply timers. A reply from a replier far from the requester is deterministically suppressed by a replier near the requester if the difference between their deterministic waiting periods is sufficiently large. Otherwise, they rely on the probabilistic wait to suppress one another.

In this paper, we investigate the relationship between the timer setting parame-

will distinguish one from the other in our later discussion. However, in SRM, a member determines its one-way propagation delay to another member by taking half of its measured round-trip delay. Therefore, SRM assumes the paths between a pair of members are symmetric.

[4]In addition to the basic timer scheduling scheme, [1] also provides a probing mechanism for random timer adaptation. We discuss it further in Section 7.

[5]SRM assumes all session members, not only the data source, may save all the application data. If some members do not save the data requested, they simply do not participate in the error recovery process.

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.,

[6] Member p is a neighbor of member q if p competes with q in sending requests or replies for a loss.

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

$$P = \frac{\int_0^{t_N} 1dx + \int_{t_N}^{B\cdot t_s} (\frac{B\cdot t_s - x + t_N}{B\cdot t_s})^{N-1} dx}{B \cdot t_s} = \frac{t_N}{B \cdot t_s} + \frac{1}{N} \cdot (1 - (\frac{t_N}{B \cdot t_s})^N) \leq \frac{t_N}{B \cdot t_s} + \frac{1}{N}$$

Since there are N members within p's requester neighborhood, the expected number of requests regarding a loss from source s is

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

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 $E' \leq \frac{n}{b} \cdot \frac{t_n}{t_p} + 1$.

3.3 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

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

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 $D + D' + 2 \cdot t_s$.

3.4 Discussion

In Section 3.3, we found that A contributes the majority of request delay when there are many members. Thus, 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. As we argue below, if B is selected properly the number of duplicate requests and the recovery delay can both be properly controlled.

Consequently, we choose $A = a = 0$ to minimize the recovery delay and rely on probabilistic suppression to minimize the number of duplicates. Since there is no deterministic suppression, all members that share the same loss are all in the requester neighborhood for that loss, i.e., they compete with one another to request retransmission. The remaining group members are in the replier neighborhood to compete for retransmission.

If $A = 0$, the expected number of requests and the recovery delay for a loss from a source s can be rewritten as,

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

$$D = \frac{B}{N+1} \cdot t_s \quad (2)$$

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 factor N in both equations will be neutralized. As a result, 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. In other words, the number of requests per loss and recovery delay are constant as functions of the session size. Similarly, if we choose $a = 0$ and $b = c \cdot n$, where c is a constant and n is the size of the replier neighborhood, we get $E' \leq \frac{t_n}{c \cdot t_p}$ and $D' \leq c \cdot t_p$. If the replier is the source in the worst case, the reply delay is $c \cdot t_s$. Therefore, the recovery delay is equal to $(C + c + 2) \cdot t_s$ on the average.

Generally speaking, the linear functions define the tradeoff between the number of duplicates per loss and recovery delay. Note that, C and c are universally identical, i.e., all members use the same linear functions in a session. However, C and c can be different, i.e.,the linear function for the request parameter can be different from the linear function for the reply parameter. The neighborhood sizes of members are different because they do not share exactly the same losses. Since the timer parameters are functions of the neighborhood sizes, they are also different for individual members.

4 A Revised SRM Timer Scheduling Scheme

Perfect request and reply suppression does not guarantee the absence of duplicate requests and replies; other factors may cause duplicates. For example, a requester may send a premature second request before the reply arrives. A premature request is not only a duplicate request but also a trigger for duplicate replies. Furthermore, if duplicate requests are sent, a replier may issue a duplicate reply if a second request arrives after the first reply has been sent. In this section, based on the conclusion in Section 3.4, we will discuss mechanisms that prevent premature requests and ignore unsuppressed requests, and propose a revised version of SRM timer scheduling scheme.

4.1 Preventing Premature Requests

After sending a request, the requester backs off its request timer and schedules a second request to ensure retransmission reliability. If the backoff timer expires before

Figure 3: Scenario of premature requests

In Figure 3, requester p sends its first request at time 0. Replier q receives the first request at t_{pq} and schedules a reply between $t_{pq} \sim t_{pq} + b \cdot t_{pq}$. If q's reply timer expires first, the waiting period of its scheduled reply is less than $3 \cdot c \cdot t_{pq}$ with a probability of 95%. The arrival time of a reply at p should be smaller than the earliest sending time of the second request. We assume p backs off it request timer for a period of $I \cdot t_{sp}$,

$$2 \cdot t_{pq} + 3 \cdot c \cdot t_{pq} \leq I \cdot t_{sp} \Rightarrow (2 + 3 \cdot c) \cdot \frac{t_{pq}}{t_{sp}} \leq I \Rightarrow I \geq 2 + 3 \cdot c$$

Note that $\frac{t_{pq}}{t_{sp}} \leq 1$ because a lost packet is recovered by the source in the worst case.

Thus we conclude that, after sending the first request, member p has to back off its request timer at least for a period of $(2 + 3 \cdot c) \cdot t_{sp}$, where c is the constant parameter used in the linear function to determine the reply parameter (e.g., b) and is universally identical for all members in a session.

4.2 Ignoring Unsuppressed Requests

Request suppression and premature request prevention reduce the chance of duplicate requests. However, the performance depends heavily on network topology,

Figure 4: Scenario of unsuppressed requests

Figure 4 shows the scenario of an unsuppressed request. Member p sends a request asking for retransmission. The request does not suppress the scheduled request in member q. Therefore, q sends its own request for the same loss and this request should be ignored by replier r. The hold period of r's scheduled replies should not be greater than the second request from p because the previous reply may be lost and the replier should be able to respond to the second request to ensure transmission reliability. We assume the hold period for r is $H \cdot t_{sp}$, where t_{sp} is the propagation delay from source s to the requester p who sent the first request. If p sent its first request at time 0, the earliest sending time of its second request is $I \cdot t_{sp}$. Therefore,

$$t_{pr} + H \cdot t_{sp} \leq I \cdot t_{sp} + t_{pr} \Rightarrow H \leq I \Rightarrow H \leq 2 + 3 \cdot c$$

After scheduling a reply triggered by the first request from p, r should ignore further requests of the same loss for a period of $(2 + 3 \cdot c) \cdot t_{sp}$. Member p should include its propagation delay from source s, t_{sp}, in its request in order for r to compute the proper hold period.

4.3 Revised Error Recovery Scheme

In this section, we will describe the revised timer scheduling scheme from SRM based on the conclusion from the previous sections.

Before a session is activated, two linear functions are selected. These linear functions are used to compute the timer setting parameters from the neighborhood sizes. For example, the request timer parameter B is defined as $B = C \cdot N$, where C is a constant and N is the requester neighborhood size; And the reply timer parameter b is defined as $b = c \cdot n$, where c is a constant and n is the replier neighborhood size. Since members have different neighborhood sizes, in the following sections we will use the notation of N_p^s [7] 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.

Members in a session exchange session messages to determine propagation delays between each pair of members and to detect data losses. When member p detects a loss from source s, it schedules a request timer between $0 \sim B_p^s \cdot t_{sp}$, where t_{sp} is the propagation delay from source s to p. When the timer expires, p multicasts it request to the session group and schedules a backoff request between $I \cdot t_{sp} \sim (I + B_p^s) \cdot t_{sp}$ to ensure retransmission reliability. If p receives a request of the same loss before its own request timer expires, it backs off its timer by rescheduling a second request between $I \cdot t_{sp} \sim (I + B_p^s) \cdot t_{sp}$. From Section 4.1 we know $I = 2 + 3 \cdot c$.

A member q with the requested data responds to the request from p by scheduling a reply between $0 \sim b_q^p \cdot t_{pq}$, where t_{pq} is the propagation delay from p to q. Furthermore, q ignores further requests for the same loss for a period of $H \cdot t_{sp}$. From Section 4.2, we know $H = 2 + 3 \cdot c$. If q receives a reply of the same loss before its reply timer expires, q simply cancels its reply. Otherwise, q multicasts its reply when its reply timer expires.

5 Dynamic Timer Adjustment

From the previous section we conclude that selecting timer parameters based on the neighborhood size produces optimal results in terms of the recovery delay and the number of duplicates. Unfortunately, members do not know their neighborhood sizes beforehand. Furthermore, there are other dynamic factors which may affect the error recovery performance. For example, network traffic load affects the propagation delay, dynamic membership change affects the neighborhood size, and network topology change affects the loss-sharing characteristic among members. Members need to adapt their requester and replier neighborhood sizes (i.e., the timer setting parameters) to these dynamic factors by learning from network feedback.

Figure 5 shows the detailed control loop of our dynamic adjustment mechanism. Each member's feedback interpreter observes network feedback to estimate neighborhood size. A parameter adjuster calculates the timer parameters from the estimated neighborhood size; And a timer scheduler schedules requests and replies with the new timer parameters. We already discussed the procedure to schedule request and reply timers in Section 4.3. We will concentrate on the procedures of feedback interpretation and parameter adjustment in the following sections.

[7] To be specific, the requester neighborhood sizes of member p are different for losses at individual lossy links from a source. However, it is impossible for a member to identify the point of loss, we use the location of the data source as an approximation.

Figure 5: Control loop of dynamic timer adjustment

5.1 Interpreting Neighborhood Size from Duplicates

A feedback interpreter observes network feedback to adjust the estimated neighborhood sizes. There are two kinds of feedback that can be observed by members for individual losses: the recovery delay and the number of requests and replies. An interpreter can estimate the neighborhood sizes based on either of them. For example, the requester neighborhood size can be interpreted based on the number of requests per loss by the relationship illustrated in Equation 1 on Page 7.

From Equation 1 we know the average number of requests in a neighborhood with N members is roughly $E = \frac{N \cdot t_N}{B \cdot t_s} + 1$. Therefore, each request with respect to the same loss represents $\frac{N}{E}$ neighbors. That is, a request from a member that is t_N away represents $\frac{B \cdot t_s}{t_N + C \cdot t_s}$ neighbors. For example, if requester q sends a request regarding a loss from source s, its request represents $\frac{B_q^s \cdot t_{sq}}{t_{pq} + C \cdot t_{sq}}$ neighbors to member p. Member q should supply B_q^s and t_{sq} in its request for p to calculate the proper number of neighbors represented by the request. Note that, q's request represents N_q^s neighbors to itself since the delay from q to itself is 0.

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. For example, if both member p and r receive a request from q, we would like q's request to represent more neighbors to r if r is farther from the source than is p. One simple way to achieve this is to weight the number of neighbors represented by q's requests by the delays from source s. In other words, the number of neighbors represented by q's request 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 q'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.

A member adds up all the numbers of neighbors represented by requests it received for the same loss. The new requester neighborhood size is calculated by using an exponential-weighted moving average with a weight ω [8] between the previous neighborhood size and the new estimation. The requester neighborhood size is adjusted for individual sources [9].

If members only receive one request for a loss, they achieve the perfect suppression result. This could result from the case in which members' neighborhood sizes are overestimated so they have a better chance to hear from one another, or from one member sending its request fast enough to suppress others. For the former case, members should reduce their estimated neighborhood sizes to probe for the minimum recovery delay. However, for the later case, there is no connection with whether the neighborhood size is overestimated. Therefore, in this second case, reducing the estimated neighborhood size does not necessarily improve the performance. Unfortunately, members can not distinguish one case from the other because of the lack of feedback (e.g., duplicate requests). In our approach, we assume the neighborhood size is overestimated if there is only one request per loss, and reduce a member's neighborhood size by a factor of δ. The value of δ determines the aggressiveness of the probing process. If δ is small, the accurate neighborhood size can be reached quickly, however, it is also more likely to underestimate the neighborhood size and cause duplicates. On the other hand, a δ close to one slows down the probing process; during the probing period the neighborhood size is overestimated and results in long recovery delay. In other words, δ is a tuning knob of the tradeoff between recovery delay and duplicates per loss.

Since the member who sent the only request for a loss is most likely the closest member to the lossy link, one could reduce the neighborhood size of the member who sent the request and keep other members' neighborhood sizes unchanged. However, the hypothesis of adaptive adjustment is that the 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 and the adaptive adjustment mechanism should be less aggressive so the estimation from the past can predict the average behavior of the future. For example, reducing p's neighborhood size by a factor of δ increases the chance that p will suppress other requests for the next loss. As a result, p will keep reducing its neighborhood size until duplicate requests are observed. Reducing p's neighborhood size aggressively based on the losses at one lossy link may affect the error recovery performance of losses at another lossy link. Therefore, we think it is prefered for all members who share the same loss with p to decrease their neighborhood sizes by a factor of δ as well.

The following algorithm is developed to adapt the requester neighborhood size to the dynamic changes in a multicast session. The neighborhood size is measured for individual losses. A measurement period starts when a loss is detected and it ends when a reply is received.

```
for each scheduled request in $p$ for source $s$
```

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

[9] From Footnote 7, we know 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. If there are multiple lossy links between source s and member p, N_p^s is the average of the neighborhood sizes of all lossy links.

```
$\sigma = 0$
for each request received from $q$ (including $p$ itself)
    $\sigma + = \frac{B_q^s \cdot t_{sq}}{t_{pq} + C \cdot t_{sq}} \cdot \frac{t_{sp}}{t_{sq}}$
if $p$ only receives one request then $\sigma = \sigma \cdot \delta$
$N_p^s = (1 - \omega) \cdot N_p^s + \omega \cdot \sigma$
```

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 4.2. 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 for the requester whose request is received first.

```
for each scheduled reply in $p$ for requester $r$
    $\sigma = 0$
    for each reply received from $q$ (including $p$ itself)
        $\sigma + = \frac{b_q^r \cdot t_{rq}}{t_{pq} + c \cdot t_{rq}} \cdot \frac{t_{rp}}{t_{rq}}$
    if $p$ only receives one reply then $\sigma = \sigma \cdot \delta$
    $n_p^r = (1 - \omega) \cdot n_p^r + \omega \cdot \sigma$
```

5.2 Interpreting Neighborhood Size from Recovery Delay

The requester neighborhood size can also be interpreted from the recovery delay by using Equation 2 on Page 7. The delay of the first expired request timer that is within a requester neighborhood of N members is $\frac{B}{N+1} \cdot t_s$ on the average. If the first expired request timer has the value $\vartheta \cdot B \cdot t_s$, 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 incoming requests and identify the smallest one as the first expired request.

To be specific, member q puts ϑ_q in its outgoing request if q schedules the request regarding a loss from source s for a period of $\vartheta_q \cdot B_q^s \cdot t_{sq}$. If member p identifies ϑ_q is the smallest one that has ever been observed, it uses ϑ_q to estimate its requester neighborhood size. Similarly, other members who share the same loss would identify ϑ_q as the smallest feedback. In order to facilitate members near the source to request retransmission first, ϑ_q is weighted by their delays from source s, for example, p weights ϑ_q by the ratio of $\frac{t_{sq}}{t_{sp}}$. The exponential-weighted moving average is also adopted. 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. Otherwise, if ϑ is converted to N and then the exponential-weighted moving average is performed, the result is divergent. The algorithm is shown below.

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

One advantage of interpreting neighborhood size from recovery delay is the ability to distinguish whether a member's neighborhood size is overestimated or whether it is sending requests fast enough to suppress others based on the feedback of its own requests. If a member's neighborhood size is overestimated, the new neighborhood size interpreted from its own ϑ should be smaller than its original estimated neighborhood size on the average. Whereas, if a member sends its request fast enough to suppress others, its new estimation should be greater than its original estimation. For example, if member p is the only member behind a lossy link, ϑ_p is equal to $\frac{1}{2}$ on the average and its new estimated neighborhood size is 1. Therefore, a member can still estimate its requester neighborhood size correctly even if it only receives the requests from itself.

The same mechanism discussed above can be applied to estimate the size of a replier neighborhood, the algorithm is shown below.

```
for each scheduled reply in $p$ for requester $r$
    $\vartheta = 1$
    for each reply received from $q$ (including $p$ itself)
        $\vartheta = \min\{\vartheta, \quad \vartheta_q \cdot \frac{t_{rq}}{t_{rp}}\}$
    $\theta_p^r = (1-\omega) \cdot \theta_p^r + \omega \cdot \vartheta$
    $n_p^r = \frac{1}{\theta_p^r} - 1$
```

5.3 Comments

Above we have described two different methods for estimating the neighborhood size. After making this estimation, a member adjusts its timer parameter using a predefined linear function. For example, if the estimated requester neighborhood size is N, a member would adjust its request timer parameter as $B = C \cdot N$, where C is the constant of the linear function. In the next section we compare these two methods of estimation. One might consider combining both neighborhood size interpretation mechanisms. For example, a member might take both estimates of neighborhood size and then use the average as the new estimated neighborhood size. However, since one interpreter does not reveal more information than the other, combining these two interpretation mechanisms does not make the neighborhood size estimation more accurate. Moreover, our results suggest that one need not resort to the additional complexity of combining the estimation algorithms to achieve good performance.

Before turning to the our simulation results, we wish to return to the issue of setting $A = 0$ by reconsidering what happens with non-zero A. As we have

Figure 6: Local minimum solution for a non-zero A in dynamic timer parameter adjustment

mentioned in Section 3.4, one major disadvantage with non-zero A is that it produces longer recovery delay than with $A = 0$. For example, Figure 6a shows the recovery delay on a tree topology with a single lossy link near the source. The deterministic wait dominates the recovery delay.

Furthermore, from Section 3.1, the requester neighborhood is proportional to $\frac{B}{A}$, so the effectiveness of deterministic suppression is proportional to $\frac{A}{B}$ as well. On the other hand, from Section 3.2, the number of duplicate requests within a requester neighborhood is proportional to $\frac{1}{B}$, so the effectiveness of probabilistic suppression is proportional to B. Therefore, the effectiveness of request suppression is not a monotonic function of either A or B. In other words, increasing B to facilitate request suppression may include more members within the requester neighborhood, which could introduce more duplicates. For example, Figure 6b shows the the results of the number of requests per loss on the same tree topology. For a non-zero A, the performance in terms of the number of requests per loss could be suboptimal due to the existence of local minima.

Since our revised timer scheduling scheme choose $A = a = 0$, the effectiveness of request suppression is proportional to B. In other words, given the network topology and membership distribution, the number of requests per loss is a monotonic function of B. Therefore, the complexity of the timer parameter adjustment is simplified significantly.

6 Simulation Results and Discussion

As in [1], the behavior of our proposed mechanisms can be most easily understood by first testing a variety of extreme scenarios before moving on to more complicated scenarios. Consequently, 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.

Each topology is populated with five different session sizes – 8, 16, 32, 64 and

128 – to examine scaling behavior. Three different linear functions for adjusting the timer setting parameters (e.g., B and b) based on estimated neighborhood size are tested, i.e., $C = c = \frac{1}{2}$, 1, and 2. We simulate the performance of our revised scheme with both neighborhood size interpretation mechanisms, i.e., the duplicate interpreter and the delay interpreter. Simulations using the session size as an approximation of the the neighborhood size are also run for comparison. Each simulation covers the error recovery activities of 2500 losses. The losses are generated by assigning an error rate on each link of the simulated topologies, and these error rates are fixed throughout a single simulation.

6.1 Topologies with a Single Lossy Link

The first set of simulations examines the performance of topologies with a single lossy link near the data source. That is, all members, except the source, share identical losses and compete to request retransmission. Therefore, the size of the session is the actual requester neighborhood size. We did not plot the results of the average number of replies per loss because the source is the only replier and the average number of replies per loss is very close to one [10]. Note that, the analysis in the following sections from the single replier scenario can be applied to the single requester scenario. That is, if only the leaf member loses packets, the average behavior of the repliers is similar to the average behavior of the requesters in the single replier scenario.

In the simulations, we choose $I = H = 2 + 3 \cdot c$, $\omega = \frac{1}{8}$, and $\delta = .90$ for the duplicate interpreter. 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.

6.1.1 Star Topology

Figure 7 shows the simulation results in the star topology. The dash curves represent the results from simulations of the duplicate interpreter, the solid curves represent the results from simulations of the delay interpreter, and the gray curves represent the results from simulations that use session size as an approximation of the neighborhood size. The average number of requests per loss and the recovery delay measured in simulations that use both feedback interpreters are very close to the results from simulations that use the session size as an approximation which suggests that the estimated neighborhood sizes from both interpreters are very similar to the actual neighborhood size (i.e., the session size). Consequently, both the average number of requests and recovery delay remain constant regardless of the session size. Note that,the average requests per loss from simulations with small C is greater than the average requests per loss from simulations with large C. On the

[10]The mechanism to ignore unsuppressed requests reduces the chance of response to a replied request. However, in some rare pathological cases, there are multiple replies sent by the source. As we have discussed in Section 4.1, the probability of premature requests is below 5% by choosing $I = 2 + 3 \cdot c$, therefore, the chance of duplicate replies is also below 5%.

Figure 8: Neighborhood size distribution in the 8-node star topology : all members share identical losses

Figure 9: Simulation results in the string topology : all members share identical losses

Figure 9 shows the simulation results in the string topology. In Figure 9(a), the number of requests per loss asymptotes to a constant [11] except for the simulations of the duplicate interpreter. As shown in Figure 10, the neighborhood sizes interpreted from duplicates are distributed in a narrower range than the estimated neighborhood sizes interpreted from delay. Hence, members are less likely to hear from one another, and as a result they generate more duplicate requests.

Furthermore, since members weight their estimated neighborhood sizes by their relative distance from the source, members near the source have smaller estimated neighborhood sizes than members far from the source. The estimated neighborhood sizes for the member closest to the source are plotted in block dot in Figure 10.

The recovery delay in Figure 9(b) decreases with the session size because the recovery delay is measured in terms of the one-way propagation delay from the source. Members far from the source rely on the requests from members near the source for retransmission without sending their own requests. Consequently, the average recovery delay is reduced when the length of the string topology increases. If a feedback interpreter is adopted, the estimated requester neighborhood size is a

[11]The number of requests is proportional to the ratio of the propagation delays among members and the propagation delay from the source, i.e., $E \leq \frac{N}{B} \cdot \frac{t_N}{t_s} + 1$. When the length of the string topology increases, the ratio of propagation delays (i.e., $\frac{t_N}{t_s}$) also increases. However, the increased amount is negligible and the number of requests per loss still remains constant.

Figure 11: Simulation results in the tree topology : all members share identical losses

Figure 11 shows the simulation results in the tree topology. Unlike the results in the star and string topologies, the number of requests per loss increases with the session size. As we have mentioned in Section 3.4, the expected number of requests per loss is $\frac{t_N}{t_s}$, where t_N is the propagation delays among members and t_s is the propagation delay from the source to members. For the tree topology, t_s is analogous to the

Figure 12: Neighborhood size distribution in the 8-node tree topology : all members share identical losses

6.1.4 Comparison among Different Topologies

Figure 13 shows the request distribution among members in the 8-node topologies. Data source is labeled as *src* and other members are labeled as m_1 through m_7 according to their distance from the source. Requests are evenly distributed among members in the star topology because all members have equal distance from the source. However, in the string and tree topologies, members near the source send more requests than members far from the source. The difference is more obvious if a feedback interpreter is adopted. Note that, m_2 and m_3 in the tree topology have similar distributions because their distances from *src* are identical.

Figure 14 shows the request and recovery delay distribution of individual losses

Figure 14: Distributions of the number of requests per loss and recovery delay in 8-node topologies : all members share identical losses

in the 8-node topologies. Since members have the same distance from the source in the star topology, one member's requests do not have a better chance to suppress others. So the number of losses triggering one request is similar to the number of losses triggering two requests. In other words, more duplicate requests are generated in the star topology. On the other hand, in the string topology, the member closest to the source has the best opportunity to request retransmission and other members can take advantage in terms of the recovery delay. Therefore, most of the losses are recovered within the round-trip time.

As we have mentioned in Section 5.1, δ is another parameter to control the tradeoff between the number of requests per loss and the recovery delay. However, the effectiveness of δ is minor. We simulated our mechanism with four different values of δ – .50, .67, .75 and .90 – and the difference of their performance is not significant. The simulation results are shown in Table 1.

Figure 15: Simulation results in the star topology : all links with uniformly-distributed error rates

In the star topology, members have independent losses so the number of requests per loss is close to one. The number of replies per loss and recovery delay are constant regardless of the size of the session. In the string topology, since members' estimated neighborhood sizes are more closely distributed in the simulations of the duplicate interpreter, they send requests and replies more aggressively. As a result, the number of request and replies per loss is higher, and the recovery delay is smaller than the simulations of the delay interpreter.

In the tree topology, we notice that the number of requests decreases with the session size and the the number of replies increases with the session size. Since the width increases exponentially with depth in the tree topology, more losses are

Figure 17: Simulation results in the tree topology : all links with uniformly-distributed error rates

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. Thus, the behavior of the repliers is very similar to the behavior of the requesters in the single lossy link case. We found Figure 17(b) is very similar to Figure 11(a).

It is the worst case scenario that all links in a topology are lossy. To understand the average behavior of our dynamic timer scheduling mechanisms, we randomly select $\frac{1}{8}$ of the links in a topology to be with uniformly-distributed error rates. In other words, there 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 Table 2 [12], and they are consistent with the results and analysis from topologies that all links are lossy.

[12]The number of requests per loss in the topologies with one lossy link is less than one, because some of the requests are queued in the network when the simulation terminates.

Table 2: Simulation results : $\frac{1}{8}$ of the randomly-selected links are with uniformly-distributed error rate

6.3 Discussion

Based on the simulation results, the delay interpreter appears to be a better choice for our dynamic timer adjustment mechanism than the duplicate interpreter. First of all, the duplicate interpreter probes for the optimal neighborhood size if there is no duplicate. Reducing the estimated neighborhood size by a factor of δ may not be the right decision in some cases. As we have seen in the simulations, reducing the estimated neighborhood size causes aggressive action in sending the requests and replies. Secondly, with the duplicate interpreter, the tradeoff between the number of duplicates per loss and the recovery delay is defined by the linear functions (parameter C and c) and the value of δ, which makes the tuning more complicated and the performance less predictable.

Finally, as we have identified in Equation 1, the number of duplicates per loss is affected by not only the neighborhood size (N), but also the neighborhood radius (t_N); In other words, the distribution of neighbors. Our revised timer scheduling mechanism can eliminate the influence of the neighborhood size, but it is unable to control for the neighborhood radius. One could propose the use of a non-zero A (and a) to control the neighborhood radius. For example, from Section 3.1, the ratio of $\frac{B}{A}$ has to be fixed in order to control the requester neighborhood radius. However, it is impractical to adjust A and B at the same pace in terms of the recovery delay. We think the best solution to control for the neighborhood radius is to localize the error recovery scope [6, 7]. Thus, the neighborhood radius is constrained by the scope of requests or replies. More research is required.

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 [8]. 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 sun the error recovery algorithm and they rely on the randomization of timers to suppress duplicate error recovery messages. A few examples are discussed below.

7.1 Structure-Based Approaches

RBP [9] 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 [10], 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 from 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.* [11] 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 [12, 13], 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 [14] 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.

7.2 Timer-Based Approaches

Grossglauser presents DTRM [15] 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.

Floyd *et al.* [1] proposed an adaptive adjustment mechanism of the random timer. A threshold of the number of duplicates per loss and a threshold of the recovery delay are predefined. Timer parameters are increased if the average number of duplicates per loss is greater than the duplicate threshold. Otherwise, the parameters are decreased if the average recovery delay is greater than the delay threshold. Therefore, the mechanism satisfies the duplicate threshold first and then the delay threshold.

8 Conclusion

We investigated the relationship between the timer setting parameters and error recovery performance in SRM. Both analysis and simulations suggest that the deterministic wait from the random timer can be removed to reduce the recovery delay and the probabilistic waiting period should be proportional to the member's neighborhood size to facilitate duplicate suppression. 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.

We revised the current timer scheduling scheme in SRM which eliminates the deterministic waiting periods of the request and reply timers, minimizes the overhead of premature requests and prevents duplicate replies in response to unsuppressed requests. We also compared two feedback interpretation mechanisms in our dynamic timer parameter adjustment. Members estimate their requester and replier neighborhood sizes from the network feedback independently to adjust their timer parameters. From the simulation results, we found that the mechanism of interpreting neighborhood size from recovery delay performs better than the mechanism of interpreting neighborhood size from duplicates.

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] 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.

[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] S. Deering. "Host extensions for IP multicasting". *Internet Draft, RFC1112.* August 1989.

[6] Ching-Gung Liu. "A Scalable Reliable Multicast Protocol". *Ph.D. Dissertation Proposal, University of Southern California.* November 1995.

[7] Ching-Gung Liu, Deborah Estrin, Scott Shenker and Lixia Zhang. "Local Error Recovery in SRM : Comparison of Two Approaches". *Technical report USC 97-648, University of Southern California.* February 1997.

[8] 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.

[9] J. Chang and N. F. Maxemchuk. "Reliable Broadcast Protocols". *IEEE/ACM Transactions on Computer Systems, Vol.2, No. 3, pp. 251-275.* August 1984.

[10] S. Armstrong, A. Freier and K. Marzullo. "Multicast Transport Protocol". *Internet Draft, RFC1301.* February 1992.

[11] 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.

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

[13] 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.*

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

[15] M. Grossglauser. " Optimal Deterministic Timeouts for Reliable Scalable Multicast". *Proceedings of IEEE INFOCOM 1996, pp. 1425-1432.* April 1996.