

Message Stability Detection for Reliable Multicast

Katherine Guo
Bell Laboratories
Lucent Technologies
Holmdel, NJ 07733
kguo@bell-labs.com

Injong Rhee
Department of Computer Science
North Carolina State University
Raleigh, NC 27695-7534
rhee@csc.ncsu.edu

Abstract—Many scalable reliable multicast protocols use the local repair scheme where certain receivers retransmit packets by other receivers. Such schemes need a mechanism, called *message stability*, to ensure reliable delivery to all members of a multicast group and to delete those packets received by all members from the buffers of the local repairers. We propose a new protocol for message stability based on random gossiping. The protocol offers scalability and fault-tolerance by limiting each of its message transmissions only to a constant number of randomly chosen group members, hence eliminating message implosion and single point failure through the diffusion of responsibility. Both statistical analysis and simulation study indicate that our gossip-style message stability protocol can be highly effective for large scale reliable multicast.

I. INTRODUCTION

The recent growth of multicast capable networks like MBONE has triggered the wide range deployment of multicast applications. Many applications that could benefit from multicasting require reliable data delivery service not provided by IP Multicast [6]. These include interactive applications such as distributed whiteboard [10], group editors, multiplayer games, distributed interactive simulation [14], and bulk file transfers. The reliable delivery service ensures that messages sent by all senders are received by all members of a multicast group.

Scalable reliable multicast protocols such as SRM [10], RMTP [21], LBRM [14], LMS [20], and Search Party [5] use the local repair scheme where certain (or all) group members, called *repairers*, retransmit packets lost by other members. These protocols typically employ a receiver-initiated NACK scheme where receivers send feedback only when packet losses occur. In the absence of positive acknowledgment, it is difficult for repairers to know when buffered packets are no longer needed for retransmission. In an environment involving many repairers, especially when repairers are randomly selected as in SRM, or regardless of available storage capacity, detecting when to delete a packet from the buffer is very critical.

Current scalable reliable multicast protocols largely ignore this issue. SRM [10] simply requires that data needed for retransmission be built from the application. RMTP [21] buffer the entire multicast session in a secondary store, but this solution can be unscalable. Search Party [5] uses a coarse timer to delete packets from memory, but does not specify how the timer is set. The only scalable solution proposed in literature is to use a shared ack tree and delete a packet from memory if all members have acknowledged reception of the packet [17]. However, complex protocols are required to build the ack tree and main-

tain it when group membership changes.

A message (we use messages and packets interchangeably) is called *stable* if it is received by all the members of the group. To safely discard messages, a mechanism is needed to detect which messages are stable. Also a failure detection mechanism is needed to report the current group membership, otherwise a failed member could prevent a message from being discarded forever.

In this paper, we propose a new mechanism based on random gossiping that detects group membership changes and collects acknowledgments to offer buffer management to existing large-scale reliable multicast protocols. The new protocol is resilient against both message losses and process failures. It is insensitive to network topology, giving consistent results regardless of the topology. It scales in detection time, as the message stability detection time increases $O(n \log n)$ with group size n . It scales in both process load and network load, as the bandwidth usage and the number of protocol messages processed during unit time increases linearly with n .

The random gossiping technique has been used in [7], [11], [23], [4] for information distribution. The contribution of our protocol is in application of gossip to detect multicast message stability, with or without membership information. In addition, our protocol can be used with any reliable multicast protocols for message stability detection. Part of Bimodal Multicast [4] uses a similar technique to detect message stability. However, it requires knowledge of group membership.

The paper proceeds as follows: Section II presents the stability detection protocol followed by analysis in Section III. Section IV proposes the modification when group membership is unknown. Section V examines the behavior of the stability detection protocol in various scenarios using simulation. Section VI discusses extensions to the basic global gossip framework using the concept of local gossip. Finally, Section VII presents conclusions.

II. STABILITY DETECTION PROTOCOL

We describe the protocol for an n -member multicast group with members numbered from 1 to n . Without loss of generality, we assume there are m ($m < n$) senders numbered 1 through m . Senders assign each data message a unique sequence number. Therefore data messages have unique names; this name consists of the global unique sender name and a local unique sequence

number.

To ensure messages are safely discarded from memory, one must detect when a message is stable and obtain a consistent view of the current group membership.

We define *ArrayMin* to be the element-wise array minimum of input arrays, and *ArrayMax* to be the element-wise array maximum. The following information is maintained at each member:

- An m -element sequence number array R where its j -th element $R[j]$ is the maximum sequence number such that all messages with less sequence numbers from sender j have arrived at this member.
- An n -element “Live” array L reflecting current group membership.
- An n -element “Timer” array T where each entry is a timer corresponding to each entry in “Live” array L .
- An n -bit “Whom-I’ve-heard-from” bitmap array W for recording from which members it has received the sequence number arrays. A message is stable if it is received by all members of the current group.
- An m -element “Min-so-far” array M which is *ArrayMin* of sequence number arrays of itself and of other members that it has heard from.
- An m -element stability array S where $S[j]$ is the minimum of the j -th element of every member’s sequence number array.

The simplest way to detect stability is for each member to send its sequence number array R to one designated member, the *coordinator*. After receiving the sequence number arrays from all members, the coordinator calculates their *ArrayMin*, and creates the stability array S . The coordinator then multicasts S in the group. After receiving S , each member can release data messages from sender j with sequence numbers less than $S[j]$.

When the group size is large, an implosion problem will occur at the coordinator, which makes the naive method not scalable. Adding a multi-level hierarchy using a shared ack tree as in [17] reduces the implosion problem but introduces new problems. One such problem appears when some interior nodes in the hierarchy crash. The chance for member crashes increases as group size increases. In a large multicast group, membership change is frequent, thereby requiring the tree to be rebuilt frequently. This fact makes the pure hierarchical approach not scalable. On the other hand, for reliable multicast protocols like LBRM [14] and RMTP [21] with built-in hierarchical structures, these structures can also be used for stability detection. Since the management of group membership hierarchies is already provided by the multicast protocols, the hierarchical extensions of the naive method can be used to improve scalability.

Stability detection. We use the random gossiping technique [3], [7] to achieve scalability and fault-tolerance. The protocol is divided into equally timed steps. During each step, every member sends its current information to b randomly chosen group members we call a *gossip subset*. The gossip subset changes from member to member and from step to step, it is therefore not efficient to assign them multicast addresses. Instead b unicasts are used to send messages to a gossip subset

of size b . In the first step, every member sends its sequence number array R to its gossip subset. After receiving a gossip message, a member sets “Min-so-far” array M to *ArrayMin* of sequence number arrays of itself and of other members that it has heard from. It also sets the “Whom-I’ve-heard-from” array W to *ArrayMax* of “Whom-I’ve-heard-from” arrays of itself and of other members that it has heard from. In the subsequent steps, every member gossips its “Min-so-far” array M and its “Whom-I’ve-heard-from” array W to different random subsets. After certain number of steps, one member receives information about all current members, and the “Min-so-far” array M at this member becomes the stability array S when the “Whom-I’ve-heard-from” bitmap array W contains all 1’s.

At this point, this member starts disseminating S in the group by putting it on the future gossip messages. Upon receiving S , a member discards stable messages accordingly. To save on bandwidth requirement of future gossip messages and to disseminate S faster, one could multicast S in the entire group.

Failure detection. However, when current group members crash or leave the group, their corresponding entries in “Whom-I’ve-heard-from” array W will stay 0, thus preventing stability detection altogether. To circumvent this, we use “Timer” array T where each entry is a timer associated with a group member. At the beginning of each round, all the timers in T are initialized to a . Whenever an entry in W changes from 0 to 1, its timer in T is reset. Whenever a timer in T expires, the corresponding member is declared to have left the group and its entry in W is set to 1. Each member independently decides the current membership based on soft state and releases stable messages accordingly. It could happen that at some members, a certain member X is declared to have left the group and comes back again in future rounds. Therefore, there could be messages that are not received by X but declared stable and discarded by these members. This would increase retransmission delay if members nearby X have all deleted messages for X ’s retransmission request. The value of timer a must be set to at least the message stability detection time.

The end of a round of the protocol is reached at each member when the member receives S . Each member keeps a round number to distinguish gossips from different rounds. The starting points of gossip for group members are scattered randomly during the interval of one step rather than concentrated at the beginning of each time step. This effectively reduces unwanted synchronization and message bursts. The pseudo-code is presented in Figure 1.

III. ESTIMATION OF STABILITY DETECTION TIME AND TIMER VALUE a

The stability detection protocol requires two input parameters — the step interval for each gossip step and the subset size for each gossip. In order to find suitable values for them, it is necessary to understand how these parameters affect the probability of an *incomplete* stability detection. A stability detection is *incomplete* if not every member has received sequence number information from all other members.

Notation:

Given arrays A and B , $A < B$ means $A[i] < B[i]$ for all i .

Each member i keeps five arrays and one number.

T_i : Timer array.

R_i : Sequence number array.

M_i : Minimum-so-far array.

W_i : Whom-I've-heard-from array.

S_i : Stability array at the end of the previous round.

r_i : Round number for the current round.

Initially, every member i has $T_i[j] = a$ for all j ,

$M_i = R_i$, $S_i = [0 \dots 0]$, $W_i[i] = 1$, $W_i[j] = 0$ for $j \neq i$ and $r_i = 0$.

Periodically each member i sends to a random gossip subset

a message containing (M, W, S, r) where $M = M_i$, $W = W_i$,
 $S = S_i$, and $r = r_i$.

Whenever $T_i[k] = 0$, set $W_i[k] = 1$.

Every member reacts to received messages as follows:

Upon receipt of a data message, member i updates R_i .

Upon receipt of a gossip message (M, W, S, r) at member i :

```

 $T_i[j] = a$ ; for all  $j$  such that  $W_i[j] = 0$  and  $W[j] = 1$ ;
if ( $r == r_i$ ) /* message is in the current round */
  if ( $W_i > W$ ) /* this message is redundant */
    do nothing;
  else if ( $W_i < W$ ) /* this message is more up-to-date */
     $M_i = M$ ;  $W_i = W$ ;
  else /* normal process */
     $M_i = \text{ArrayMin}(M_i, M)$ ;  $W_i = \text{ArrayMax}(W_i, W)$ ;
  end if
 $S_i = \text{ArrayMax}(S_i, S)$ ; /* each round can have up to  $n$ 
concurrent  $S_i$ 's, the max is the most up-to-date one. */
if ( $W_i$  contains all 1's) /* start next round */
   $T_i[j] = a$  for all  $j$ ;
   $S_i = M_i$ ;  $r_i = r_i + 1$ ;  $M_i = R_i$ ;  $W_i[i] = 1$ ;  $W_i[j] = 0$  for  $j \neq i$ ;
end if
else if ( $r == r_i + 1$ ) /* message is from the next round */
   $M_i = M$ ;  $W_i = W$ ;  $S_i = \text{ArrayMax}(S, S_i)$ ;  $r_i = r$ ;
  if ( $W_i$  contains all 1's) /* start another round */
     $S_i = \text{ArrayMax}(S_i, M_i)$ ;  $r_i = r_i + 1$ ;  $M_i = R_i$ ;
     $W_i[i] = 1$ ;  $W_i[j] = 0$  for  $j \neq i$ ;
  end if
else if ( $r > r_i + 1$ )
  error;
else if ( $r < r_i$ ) /* message is from previous rounds */
  do nothing;
end if

```

Fig. 1. Pseudo-code for Gossip. In this version, the stability array S is piggy-backed on all gossip messages.

The goal of the gossip-style protocol is to disseminate information in the group step by step, where during each step, each member sends information it has collected to a random subset of the group. We follow the analysis used in epidemic theory [2], [4], [7], [11] given that the execution is broken up into synchronous steps, during which every process gossips once. For a particular sender i , a member that has received the sequence number array for sender i is called *infected*. It is indicated by $W[i] = 1$ in the ‘‘Whom-I’ve-heard-from’’ array W . During each step, the probability that a certain number of members are infected given the number of already infected members is calculated. This is done for different number of infected members. Therefore at the end, the number of steps needed to achieve a certain probability that every member is infected can be found.

In practice, each member gossips at regular step intervals, but the intervals are not synchronized. In fact, we intentionally scatter the gossips from different members in each step to

reduce message burstiness. Message propagation time is typically much shorter than the length of step intervals. We define *micro-step* to be a step in which only one random member is gossiping. Note this member is not necessarily infected. The member then chooses a number of other members to gossip to at random. This set of members constitute a gossip subset of size b . Thus, in a micro-step, the number of infected members can grow by at most b .

A. Analysis when the size of gossip subset is one

To simplify the analysis, we start with $b = 1$. As a first step, assume only one member’s sequence number array needs to be disseminated in the group. Let n be the total number of members, k_i be the number of infected members in micro-step i , and P_a be the probability that a gossip successfully arrives at a chosen member before the start of the next micro-step. Only one member is infected initially. The number of infected members cannot shrink because we assume group membership remains constant during the execution of the stability detection protocol.

We model the gossip process as a discrete time Markov chain. The state of the system denotes the number of infected members. State transitions occur at the end of each gossip micro-step. We conduct a transient analysis of this Markov chain in which $P(k_{i+1} = k)$ is used to denote the probability that the total number of infected members becomes k after micro-step $i + 1$.

Given k out of n members are infected, the probability that the number of infected members increases by one in a micro-step is

$$P_{inc}(k) = \binom{k}{n} \left(\frac{n-k}{n-1} \right) P_a$$

And the probability that the number of infected members becomes k in micro-step $i + 1$ is

$$P(k_{i+1} = k) = (1 - P_{inc}(k))P(k_i = k) + P_{inc}(k-1)P(k_i = k-1)$$

with the following initial condition: $P(k_i = 0) = 0$, $P(k_0 = 1) = 1$, and $P(k_0 = k) = 0$ for $k \neq 1$.

The probability that any member does not get infected by the sequence number information from member X after r micro-steps is $P_{incomp}(X, r) = 1 - P(k_r = n)$. However, for any two members X and Y , $P_{incomp}(X, r)$ and $P_{incomp}(Y, r)$ are not independent. $P_{incomp}(r)$, the probability that any member is not infected by sequence number information from any other member can be bound using the Inclusion-Exclusion Principle [16]: $Pr(\cup_i A_i) \leq \sum_i Pr(A_i)$.

$$P_{incomp}(r) \leq nP_{incomp}(X, r) = n(1 - P(k_r = n))$$

From this bound we calculate how many micro-steps are needed to achieve stability detection. The calculation is done iteratively starting at the first micro-step.

B. Analysis when the size of gossip subset is three

When the size of gossip subset is greater than one, the analysis is more complicated. Here we present the case for $b = 3$, and analysis for other values of b can be conducted in a similar fashion.

Again, we assume only the sequence number information from one member needs to propagate and only one member is infected initially. Each gossip message is sent from one source to three *different* destinations.

If k out of n members are infected already, then the probability that a gossip is from an infected member is $\frac{k}{n}$.

The following are the three scenarios where the number of infected members increases by one in a micro-step:

1. One copy of the gossip message is sent to an uninfected member, two copies are sent to two different already infected members and all copies arrive successfully at their destinations. The process can be divided into two stages. In the first stage, three out of three messages arrive successfully. In the second stage, one message is sent to an uninfected member, two are sent to already infected members. The probability for this event to happen is

$$\begin{aligned} P_1 &= \binom{k}{n} \binom{3}{3} P_a^3 \frac{\binom{n-k}{1} \binom{k-1}{2}}{\binom{n-1}{3}} \\ &= 3 \binom{k}{n} \binom{n-k}{n-1} \binom{k-1}{n-2} \binom{k-2}{n-3} P_a^3 \end{aligned}$$

2. One copy of the gossip message arrives at an uninfected member, another copy arrives at an infected member and yet another copy is lost. The probability for this event is

$$\begin{aligned} P_2 &= \binom{k}{n} \binom{3}{2} P_a^2 (1 - P_a) \frac{\binom{n-k}{1} \binom{k-1}{1}}{\binom{n-1}{2}} \\ &= 6 \binom{k}{n} \binom{n-k}{n-1} \binom{k-1}{n-2} P_a^2 (1 - P_a) \end{aligned}$$

3. One copy of the gossip message arrives at an uninfected member, the other two copies are lost. The probability for this to happen is

$$\begin{aligned} P_3 &= \binom{k}{n} \binom{3}{1} P_a (1 - P_a)^2 \frac{\binom{n-k}{1}}{\binom{n-1}{1}} \\ &= 3 \binom{k}{n} \binom{n-k}{n-1} P_a (1 - P_a)^2 \end{aligned}$$

The probability that the number of infected members is incremented by one in a micro-step is then

$$P_{inc}(k, 1) = P_1 + P_2 + P_3$$

There are two cases where the number of infected members can increase by two:

1. Two copies of the gossip message are sent to different uninfected members. The other copy is sent to an infected member. None of these copies are lost. This event's probability is

$$\begin{aligned} P_4 &= \frac{k}{n} \binom{3}{3} P_a^3 \frac{\binom{n-k}{2} \binom{k-1}{1}}{\binom{n-1}{3}} \\ &= 3 \binom{k}{n} \binom{n-k}{n-1} \binom{n-k-1}{n-2} \binom{k-1}{n-3} P_a^3 \end{aligned}$$

2. Two copies of the gossip message arrive at different uninfected members successfully, and the other copy is lost. The probability is

$$\begin{aligned} P_5 &= \binom{k}{n} \binom{3}{2} P_a^2 (1 - P_a) \frac{\binom{n-k}{2}}{\binom{n-1}{2}} \\ &= 3 \binom{k}{n} \binom{n-k}{n-1} \binom{n-k-1}{n-2} P_a^2 (1 - P_a) \end{aligned}$$

Therefore, the probability that the number of infected members increases by two is

$$P_{inc}(k, 2) = P_4 + P_5$$

There is only one scenario where the number of infected members increases by three, which is when all three copies of the gossip message arrive at three different uninfected members successfully. The probability for this is

$$\begin{aligned} P_{inc}(k, 3) &= \binom{k}{n} \binom{3}{3} P_a^3 \frac{\binom{n-k}{3}}{\binom{n-1}{3}} \\ &= \binom{k}{n} \binom{n-k}{n-1} \binom{n-k-1}{n-2} \binom{n-k-2}{n-3} P_a^3 \end{aligned}$$

The probability that the number of infected members in micro-step $i + 1$ is k ($0 < k \leq n$) consists of four parts: the first part comes from the case where the number of infected members in micro-step i is k and does not increase, the second part comes from the case where the number of infected members is $k - 1$ and increases by one, and so on. This probability is

$$\begin{aligned} P(k_{i+1} = k) &= (1 - P_{inc}(k, 1) - P_{inc}(k, 2) \\ &\quad - P_{inc}(k, 3)) P(k_i = k) \\ &\quad + P_{inc}(k - 1, 1) P(k_i = k - 1) \\ &\quad + P_{inc}(k - 2, 2) P(k_i = k - 2) \\ &\quad + P_{inc}(k - 3, 3) P(k_i = k - 3) \end{aligned}$$

(when $k \geq 3$).

There are two special cases for this probability when $k = 1$ and $k = 2$. Since initially one member is infected, the probability that one member is infected in micro-step $i + 1$ comes only from the case where no member gets infected in that micro-step. Thus, $P(k_{i+1} = 1) = (1 - P_{inc}(1, 1) - P_{inc}(1, 2) - P_{inc}(1, 3))P(k_i = 1)$. Similarly, the probability that two members are infected in micro-step $i + 1$ comes from either of the two cases in which no member gets infected or exactly one member gets infected. The probability is

$$P(k_{i+1} = 2) = (1 - P_{inc}(2, 1) - P_{inc}(2, 2) - P_{inc}(2, 3))P(k_i = 2) + P_{inc}(1, 1)P(k_i = 1)$$

Since initially one member is infected, the initial conditions for this Markov chain are the following: $P(k_i = 0) = 0$, $P(k_0 = 1) = 1$, and $P(k_0 = k) = 0$ for $k \neq 1$.

$P(k_r = n)$ is the probability that all the n members get infected after r micro-steps. As with the subset size 1 case, the probability that any member does not get infected by the sequence number information from member X after r micro-steps is $P_{incomp}(X, r) = 1 - P(k_r = n)$. $P_{incomp}(r)$ is bounded by

$$P_{incomp}(r) \leq nP_{incomp}(X, r) = n(1 - P(k_r = n))$$

C. Comparing results from the analysis

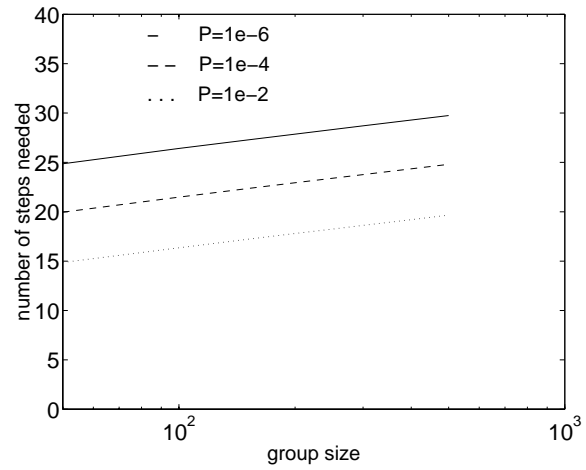
In actual execution of the protocol, every member gossips once during each step interval. Therefore the number of steps needed is the number of micro-steps divided by the group size. From the results plotted in Figure 2 in log scale, we can see clearly a $O(\log n)$ increase of number of steps with group size n . This means the number of micro-steps is on the order of $O(n \log n)$.

As shown in Figure 2, with a fixed group size, the increase of subset size decreases the number of steps needed to detect stability. When the subset size changes from 1 to 3, this decrease is only about 55%. But the number of messages sent out in the system per step interval increases three times! This analysis suggests that it is more beneficial to use a small subset size. It is better to gossip to one member 55% faster to get the same time and quality as gossiping to three members during each gossip.

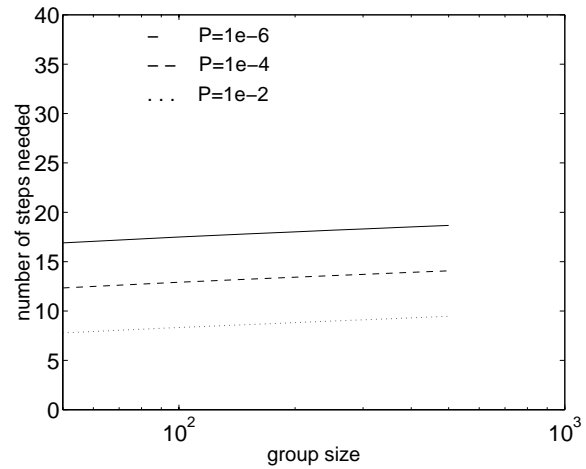
There is a trade-off between minimizing the probability of making incomplete stability detection, and the number of steps needed. Figure 3 shows this trade-off is satisfactory, in that only a few extra gossip steps are necessary for better quality.

IV. MODIFICATION WHEN GROUP MEMBERSHIP IS UNKNOWN

Most reliable multicast protocols follow the IP multicast model that receivers join and leave the multicast group without notifying senders [10], [20], [5]. In this model, none of the members have the complete group membership information. Reliable multicast protocols such as LMS [20] and Search Party [5] utilize the underlying IP multicast routing tree to conduct retransmission.



(A): subset size = 1

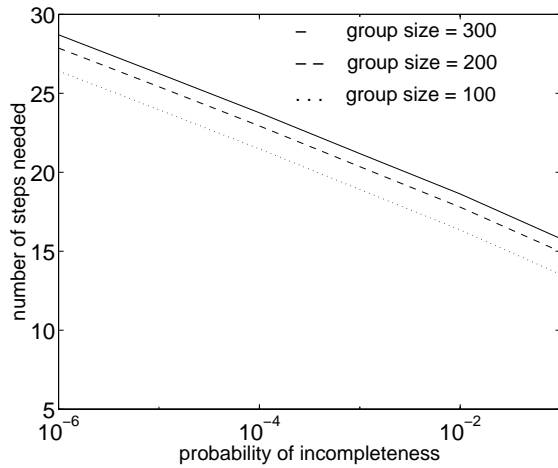


(B): subset size = 3

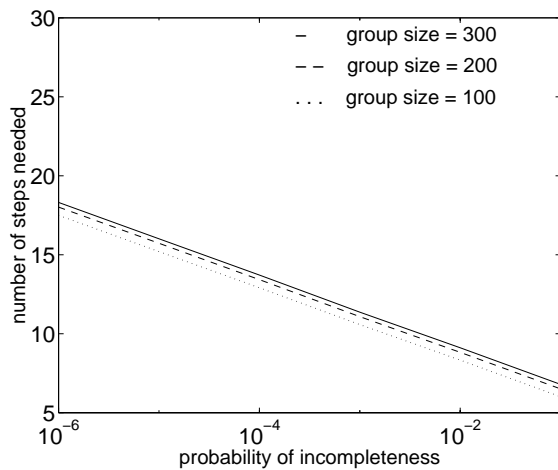
Fig. 2. The number of steps needed to achieve different probability of incomplete stability detections. P stands for P_{incomp} in the figure.

The gossip-style stability detection protocol presented in this paper can also operate without the knowledge of group membership. Two steps in the protocol require membership information: one is when a gossip message is sent to a random member, the other is when a gossip message is received where the “Whom-I-have-heard-from” array needs to be updated.

Without group membership information, we can still send a gossip message to a random member using a protocol similar to Search Party [5]. Each gossip message is sent to the group address and forwarded along the multicast tree. When a router receives such a message, it forwards the message to one of its children randomly. This means the router randomly forwards the message to an interface that is not where the message arrives from. Specifically, if a router has c children, then the probability that the message is forwarded to each child is $1/c$. However, the probability that group members are chosen to be the random



(A): subset size = 1



(B): subset size = 3

Fig. 3. Cost of quality in terms of number of steps.

destination of the message is not uniform. In a simple example, if the root router has 2 children, and the left child is connected to 1 receiver and the right child connected to 9 receivers, then 50% of the time a gossip message is sent to the single receiver on the left branch of the tree! This biased distribution of gossip messages would increase the stability detection time.

Some multicast routing protocols require routers to keep count of their children in the tree and exchange this information with other routers. For example, the EXPRESS multicast service [13] suggests that each router keep track of the number of group members in each subtree rooted at the router. With this information, a gossip message can be forwarded to a random member with uniform distribution as follows. Assume a router has c children labeled with $1, 2, \dots, c$, and $S(i)$ is the member population in the subtree rooted at child i , then the probability a gossip message is forwarded to child i is $S(i) / \sum_{j=1, \dots, c} S(j)$.

When the subset size of each gossip is $b > 1$, then one gossip message is sent out b times. This is equivalent to randomly choosing b members from the set of all members each with uniform distribution.

Without group membership information, it is impossible to keep track of “Whom-I-have-heard-from” array. Therefore, based on the calculation in Section III, we use the number of gossip steps to estimate when message stability detection is complete. With an estimate of the size of the group n , and a given probability of incomplete stability detection, we can find the corresponding number of steps needed from Figure 2.

V. SIMULATIONS OF THE STABILITY DETECTION PROTOCOL

For a given network topology, set of group members, set of senders, and patterns for message sending and message loss, it is possible to analyze the behavior of the stability detection protocol with a fixed step interval, and a fixed subset size. However, this paper focuses on the performance of the protocol across a wide range of network topologies and scenarios where group membership remains constant. For this, we conducted simulations using the ns [18] simulator.

For a large group size, the probability that a particular node in a randomly labeled tree has a degree of at most four approaches 0.98 [19], therefore, the underlying network used in the simulation is a 1000-node balanced bounded-degree tree where interior nodes all have degree four. Links in the network are bi-directional and each direction has a bandwidth of $30K$ bps allocated the messages used for stability detection. Message propagation delay on each link is $w = 5$ milliseconds, which is typical for wide area links. A rate-controlled network is assumed in which the data source is shaping its traffic by delaying packet sends to meet the $30K$ bps allocated rate requirement. Under this assumption, the expected time a u -byte message spends on the wire to travel one link is $t_0 = w + u/v$, where v is the bandwidth. The router processing time for a message is 1 millisecond. The time needed for a host to send a message follows the formula $t_s(u) = 100 + 2(94 + 35u/4000 + 50u/1000) + 50 = 338 + 47u/400$ (microseconds) [1], [15]. The time needed for a host to receive a message is normally about 10% higher than the sending time since interrupts need to be handled [1]. It is set to $t_r(u) = 1.1 \times t_s(u)$. The queuing delays incurred at the hosts and routers are also simulated by ns. The message header size is set to $h = 32$ bytes which is enough for most transport protocols [1], [22].

A. Performance indices

The most important goal for a message stability detection protocol is to minimize the time to stabilize a message, reducing the buffer space required for data messages at each member to a minimum.

To measure time requirement, we define *Time Per Round* (TPR) as the duration of time between the start of the stability detection protocol and the moment the first member constructs the stability array S . After the first member constructs S , it immediately multicasts the array in the group, therefore every

member will receive the stability information within one multicast. A more important time index is *Time-To-Stable* (TTS), which is the time between the moment a data message is multicast and the moment it is detected to be stable. TTS depends on three things: TPR, the frequency to trigger each round of stability detection, and the underlying reliable multicast protocol. Analysis of TTS requires implementation or simulation of the reliable multicast protocols. If the reliable multicast protocol can deliver a message to all the receivers within D seconds, the stability detection protocol is triggered every F seconds, and it can detect the message's stability within TPR seconds, then the maximum TTS becomes $D + F + \text{TPR}$ seconds. This means that at most $D + F + \text{TPR}$ seconds after a message is multicast from a sender, it can be deleted from the network. Since TPR is the factor that is determined by the stability detection protocol, this paper only studies TPR.

To investigate the behavior of the protocol in detail, we measure the number of steps needed for a round, and the average number of messages sent out by each member during unit time. This is an indicator of the load the protocol adds to the network.

In most applications where the multicast group is large, normally a small percent of the members are active sources for data messages. The number of senders (m) is set to 50, while group sizes range from 50 to 500. In the gossip protocol, each message used to detect stability contains a 32-byte header, an m -element sequence number array M where each sequence number occupies 4 bytes, an n -element bitmap array W where each element is one bit and an integer round number which has 4 bytes. The overall gossip message size for a group of size n is $32 + 4 \times m + n/8 + 4 = 236 + n/8$ (with some padding to make it aligned in the packet). In the simulation, n ranges from 50 to 500; and the packet size ranges from 243 to 299 bytes. Since a typical WAN can handle packets shorter than 500 bytes without fragmentation, packet fragmentation is not considered in the simulations.

For a given group size and a given number of senders, there are two control parameters – the step interval and the subset size for each gossip. We vary step interval from 1 to 20 seconds and vary subset size from 1 to 5. In a WAN, IP-multicast is not efficient in sending messages to small groups that are constantly changing [9], thus b unicasts are used to send gossip messages to each subset of size b .

B. Simulations with a fixed group size $n = 200$

For the fixed group size $n = 200$ with $m = 50$ senders, we randomly choose 200 nodes in the 1000-node tree to be members of the multicast group. Every node in the tree only can store up to 64 gossip messages. Gossip messages arriving at a node with a full buffer are dropped. Additionally, two random gossip messages in each step interval are dropped at the senders.

For each pair of subset size and step interval, the average results of 20 runs are reported. Each run contains a different randomly constructed 200-member group.

We noticed about 0.15% of the simulation runs did not detect message stability within 10 minutes. These bad samples are

excluded from the analysis in the rest of the section. The probability for a round not to finish after a relatively long time exists, but is very slim as reported in Section III. In practice, a second round of the stability detection protocol can start with a different seed for generating subsets. The probability that both rounds take an unreasonably long time is even slimmer – 0.0225%.

Since gossip messages are sent out by unicast, the number of messages sent out by each member during each step is the same as the subset size. For a given data point (x, y) in the simulation with subset size x and step interval y , the number of messages each member sends out per unit time is x/y . Therefore the traffic load generated by the protocol is proportional to x/y .

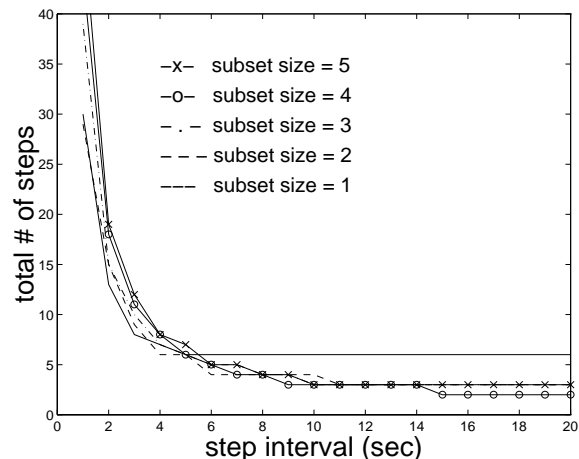


Fig. 4. The number of steps needed in a round with group size 200.

Figure 4 shows that as the step interval increases, the number of step decreases to a minimum and remains there. We call the step interval that results in the minimum number of steps the *critical point*.

For a given subset size, before reaching the critical point, a decrease in the step interval results in an increase in the number of steps. As the step interval decreases, each member is scheduled to gossip in a shorter time period. This shorter time period prevents each member from receiving all the available new information. Therefore, more steps are needed to detect message stability. Moreover, the less new information in each step, the more redundant or repeating information flows in the network, increasing network traffic load. These factors contribute to the increase in the number of steps needed for stability detection.

After the critical point is reached for each curve, gossip messages are scattered far enough to minimize both traffic load and redundant messages. In this situation, the larger the subset size, the more information is exchanged in each step, and a smaller number of steps are needed to reach message stability.

The behavior of the TPR curves plotted in Figure 5 can be derived from Figure 4, because TPR is the product of the number of steps and the step interval. Before reaching the critical point, the steeply declining trend of number of steps dominates the behavior of TPR, even though the increasing step interval

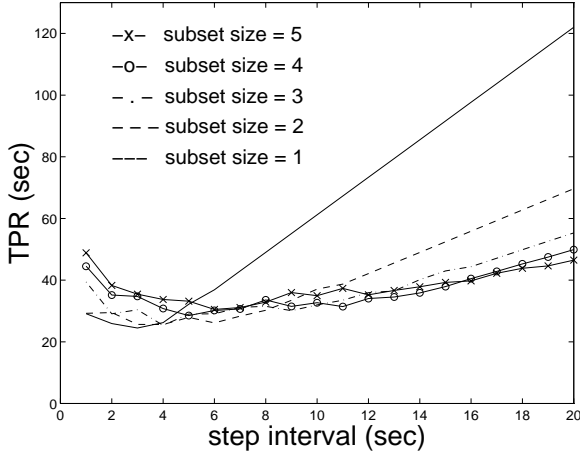


Fig. 5. Time Per Round with group size 200.

damps the TPR's decline. After passing the critical point, TPR becomes a linear function of step interval with the coefficient being the value of the number of steps, which is almost constant. As Figure 4 shows, the smaller the subset size, the larger the number of steps needed for detecting stability when the step interval passes the critical point. This feature transferred into Figure 5 says that the smaller the subset size, the steeper the slope of the TPR function is. When the step interval falls in the range between 4 and 12 seconds, the opposite movements of the two components of the TPR function cause TPR to fluctuate in a narrow range from 28 to 38 seconds, except for the case of subset size one. For each subset size, a window of optimal step intervals exists in which TPR is near-minimum.

More simulation experiments revealed that location of group members and network topology do not have a noticeable effect on the protocol. This is the advantage of gossip style protocol.

C. Adaptive method: finding the window of optimal step intervals

As observed from Figure 5, every TPR curve has a plateau portion in which one can select any step interval to achieve a near-minimum TPR. The process of finding the plateau has two parts. The first part finds a step interval that achieves a near-minimum TPR, and the second part finds a window around that step interval. If TPR can be expressed as a function, then Newton's method [8] can find its minimum. Otherwise, better approaches exist in experimental optimization. One such approach is the *golden-ratio method* [12]. Figure 6 presents the two-part algorithm for finding the window of optimal step intervals.

Different groups and network topology require different input values for the algorithm. For example, the following input values are given for the simulation in Section V-B: $g = 2$ seconds, $a_0 = 0$, $b_0 = 20$ seconds, $e = 10\%$, and $p = 1$ second. For the TPR curve for subset size of 3, part one finishes after 5 iterations, ending up with 9 seconds as the step interval that achieves a minimum TPR of 28.6 seconds. Part two finds the

lower bound $s_1 = 4$ seconds and the upper bound $s_2 = 10$ seconds after total of 5 iterations of the while loops. Any step interval in the range from 4 to 10 seconds can be used to achieve a TPR between 28.6 to 31.2 seconds. The window sizes are different for different subset sizes. As the group size and network condition change over time, this two-part algorithm is executed periodically to find the current window of optimal step intervals.

As observed from Figure 5, as the subset size increases, the minimum TPR increases slightly, but the window size of optimal step intervals increases significantly. There is a trade-off between the stability of the protocol and the minimum TPR. If the window size is too small, for example, when subset size is 1, then a slight perturbation of the network condition will result in dramatic increase of TPR if the step interval is unchanged. A large window size is preferred because slight changes in network condition will result in overlapping between the new and current windows, therefore only slight changes in TPR. As a result, the recommended subset size is 2 or 3.

Part I:

Input: a_0 : smallest step interval in the search.

b_0 : largest step interval in the search.

g : distance between two ends when the search stops.

Output: s : step interval that achieves a near-minimum TPR.

```

a := a0; b := b0; stop := false;
while ¬stop do
  d := b - a;
  if (d > g) then
    m1 := a + 0.382 * d; m2 := a + 0.618 * d;
    if (f(m1) ≥ f(m2)) then a := m1; else b := m2;
  else
    stop := true; s := (a + b) / 2;

```

Part II:

Input: s : step interval that achieves a near-min TPR.

$f(s)$: the near-min TPR from part I.

e : fluctuation index. Step intervals that achieve TPR in the range from $(1 - e) * f(s)$ to $(1 + e) * f(s)$ should be in the window.

p : initial search step size.

Output: s_1 : lower bound of the window.

s_2 : upper bound of the window.

```

s1 := s; k := 1;
while ((s1 - k * p > 0) ∧ (|f(s1 - k * p) - f(s)| < e)) do
  s1 := s1 - k * p; k := 2 * k;
k := k / 2;
while ((k ≥ 1) ∧ (|f(s1 - k * p) - f(s)| < e)) do
  s1 := s1 - k * p; k := k / 2;

s2 := s; k := 1;
while (|f(s2 + k * p) - f(s)| < e) do
  s2 := s2 + k * p; k := 2 * k;
k := k / 2;
while ((k ≥ 1) ∧ (|f(s2 + k * p) - f(s)| < e)) do
  s2 := s2 + k * p; k := k / 2;

```

Fig. 6. The adaptive algorithm: finding a near-minimum TPR. $f(x)$ is the average TPR for a step interval x .

D. Simulations with different group sizes

For various group sizes, the same pattern of simulation result is observed as for group size 200. The near-minimum TPR and

the corresponding number of steps are presented in Figures 7 and 8.

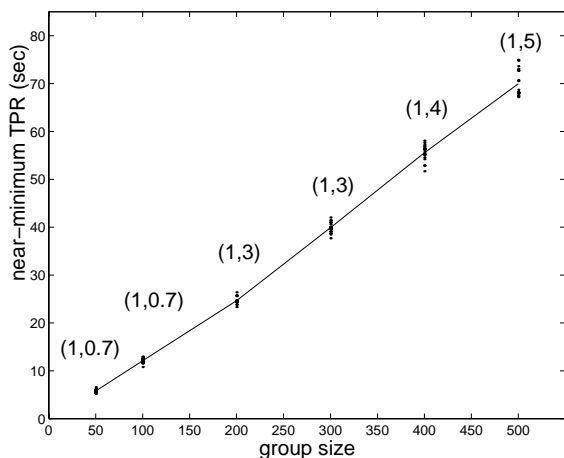


Fig. 7. Near-minimum TPR with varying group sizes. A data point with subset size x and step interval y seconds is labeled as (x, y) .

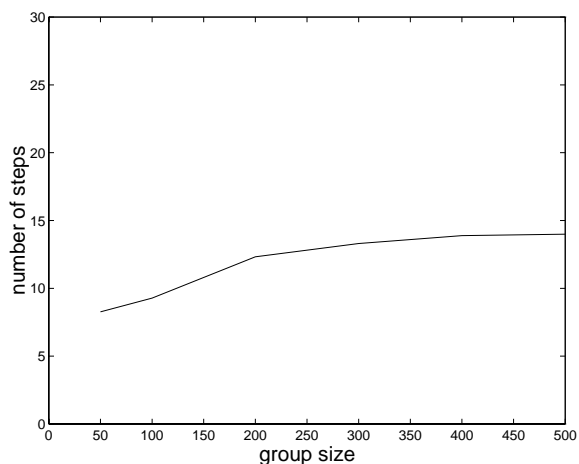


Fig. 8. The number of steps needed in a round with varying group sizes.

The group sizes range from 50 to 500. For each group size, 20 simulation runs are conducted with subset size of 1 and an optimal step interval. Each simulation is represented by a dot in Figure 7. The solid line represents the mean of the TPRs. We see the average number of steps increases with $O(\log n)$, the optimal step interval increases approximately with $O(n)$, thus the minimum TPR shows a $O(n \log n)$ increase. This is not surprising because as shown in Section III, when there is no bandwidth constraint, the gossip message from each group member reaches another member during each gossip step, and the number of steps increases with $O(\log n)$. When there is such a constraint, the step interval needs to be increased with $O(n)$ in order to keep bandwidth requirement constant and keep message burstiness to a minimum.

When network bandwidth is increased to a 300K bps, we also

observed a 10 time decrease in TPR. More bandwidth combined with an optimal step interval will result in a faster stability detection time.

VI. EXTENDING THE BASIC APPROACH – LOCAL GOSSIP

With the global stability detection framework described above, each group member on a subnet propagates its sequence number information by sending gossip messages to some random remote members. A more efficient way would be to combine and compress their information first into one sequence number array of the local group, and then allow one or several designated members to send this array to remote members.

We propose a hierarchical scheme to apply the gossip-style protocol. The multicast group is divided into a number of local groups according to their location on the network. Since many reliable multicast protocols employ built-in local groups [14], [21], their group division can be used. Each local group has G *Stability Controllers* (SCs) where G is a user-defined parameter which determines how robust the protocol is in case of SC failures. The protocol proceeds in two phases. In the first phase, each member gossips to other members in its local group trying to obtain stability information within the local group. After the local stability arrays are constructed, the SCs start the second phase by gossiping among all the SCs. After one SC receives the stability information from SCs representing the other local groups, the global stability array is constructed and multicast to the entire group.

To show the effectiveness of the local gossip scheme, we divide each group randomly into $n/50$ local groups of size 50, and in each local group, 5 SCs are randomly chosen. For gossiping in each local group of size 50, subset size of 1 and an optimal step interval of 700 milliseconds are chosen to achieve a near-minimum TPR. When n is 100, 200, 300, 400, and 500, the size of the SC group is 10, 20, 30, 40, and 50 respectively, and the optimal step interval is 200, 300, 500, 600, and 700 milliseconds respectively. The near-minimum TPR achieved by the local gossip scheme is plotted in Figure 9. Compared with the global scheme, the hierarchical scheme dramatically reduces TPR.

This significant improvement in performance comes from two factors. One, a hierarchy is built in the communication structure. Two, stability information is combined and compressed; all the stability arrays of members from each local group are combined into one stability array of the same size, effectively reducing the traffic load. This technique applies to any information gathering protocol to improve performance and scalability.

Ideally, the local groups should be constructed based on the location of the group members in the network and the SCs should be selected dynamically based on the load of group members and network topology. As a result, the performance of the protocol can be improved. We are currently investigating this issue.

The global scheme requires group membership information at every member. This is not likely to be feasible in a WAN. The local gossip scheme solves this problem by only requiring each

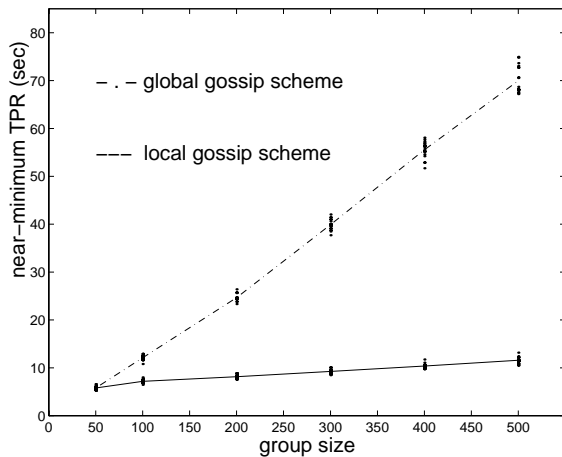


Fig. 9. Near-minimum TPR for sparse groups using the local gossip scheme. A data point with subset size x and step interval y seconds is labeled as (x, y) .

local group member to maintain the addresses of other members on the same subnet.

VII. CONCLUSION

The gossip-style stability detection protocol achieves fault tolerance and scalability. It tolerates message losses without requiring a reliable multicast protocol underneath, by periodically sending gossip messages to random sets of group members. This scheme overcomes routing errors, transient link failures and omission failures, because messages are randomly sent to other members, and a message may take a different route in different steps.

It tolerates group membership changes caused by member crashes or leaves. Although the current group membership information is maintained at each member for the failure detection protocol, agreement and instant update of group membership is not required. A similar technique is used in routing table updates. It is necessary to propagate route changes to all the routers without shutting down the network.

The scalability of the protocol comes from three features. First, the state information maintained at each multicast participant is minimal. Each member keeps the current group membership, the current round number, an m -element sequence number array, an m -element “Min-so-far” array, and an n -element “Whom-I’ve-heard-from” bitmap array. Second, the message size does not increase linearly with the group size. As shown in the simulations, the majority of a gossip message is occupied by the “Min-so-far” arrays of size proportional to the number of senders. Only a small portion is a bitmap array which is of the size of the group size n in bits. (When $n = 1000$, it occupies 125 bytes.) Commonly, the number of senders is much less than the group size, thus the message size does not increase linearly with group size. Third, no hot spot exists in the protocol, because no member receives a lot of messages in a short amount of time. The protocol is completely free of the implosion problem.

ACKNOWLEDGMENTS

We would like to thank Mark Hayden, Werner Vogels, Robber van Renesse and Ken Birman for extensive discussions and valuable suggestions.

REFERENCES

- [1] R. Ahuja, S. Keshav, and H. Saran. Design, implementation, and performance of a native mode ATM transport layer. *IEEE/ACM Transactions on Networking*, 4(4):502–515, August 1996.
- [2] N. T. J. Bailey. *The Mathematical Theory of Infectious Diseases and its Applications (second edition)*. Hafner Press, 1975.
- [3] B. Baker and R. Shostak. Gossips and telephones. *Discrete Mathematics*, 2(3):191–193, June 1972.
- [4] K. P. Birman, M. Hayden, O. Ozkasap, Z. Xian, M. Budiu, and Y. Minski. Bimodal multicast. *ACM Transactions on Computer Systems*, May 1999.
- [5] A. Costello and S. McCanne. Search party: Using randomcast for reliable multicast with local recovery. In *Proceedings of IEEE INFOCOM’99*, New York, NY, March 1999.
- [6] S. E. Deering and D. R. Cheriton. Multicast routing in datagram internetworks and extended LANs. *ACM Transactions on Computer Systems*, 8(2):85–110, May 1990.
- [7] A. Demers, D. Greene, C. Hauser, W. Irish, J. Larson, S. Shenker, H. Sturgis, D. Swinehart, and D. Terry. Epidemic algorithms for replicated database maintenance. In *Proceedings of the Sixth ACM Symposium on Principles of Distributed Computing*, pages 1–12, Vancouver, British Columbia, August 1987.
- [8] J. E. Dennis, Jr. and R. B. Schnabel. *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*. Prentice Hall Series in Computational Mathematics. Prentice-Hall, Inc., 1993.
- [9] M. Doar and I. Leslie. How bad is naïve multicast routing? In *Proceedings of IEEE INFOCOM’93*, pages 82–89, San Francisco, CA, March 1993.
- [10] S. Floyd, V. Jacobson, C. Liu, S. McCanne, and L. Zhang. A reliable multicast framework for light-weight sessions and application level framing. *IEEE/ACM Transactions on Networking*, November 1996.
- [11] R. Golding and K. Taylor. Group membership in the epidemic style. Technical Report UCSC-CRL-92-13, UC Santa Cruz, Department of Computer Science, May 1992.
- [12] *Handbook of Mathematics*. High Education Publishing House, Beijing, 1979.
- [13] H. Holbrook and D. Cheriton. IP multicast channels: EXPRESS support for large-scale single-source applications. In *Proceedings of ACM SIGCOMM’99*, September 1999.
- [14] H. Holbrook, S. Singhal, and D. Cheriton. Log-based receiver-reliable multicast for distributed interactive simulation. In *Proceedings of ACM SIGCOMM’95*, pages 328–341, Cambridge, MA, August 1995.
- [15] J. Kay and J. Pasquale. The importance of non-data touching processing overheads in TCP/IP. In *Proceedings of ACM SIGCOMM’93*, San Francisco, CA, September 1993.
- [16] D. Kozen. *The Design and Analysis of Algorithms*. Springer Verlag, 1991.
- [17] B. Levine, D. Lavo, and J.J. Garcia-Luna-Aceves. The case for reliable concurrent multicasting using shared ack trees. In *Proceedings of ACM Multimedia’96*, pages 365–376, November 1996.
- [18] S. McCanne and S. Floyd. Ns (network simulator). Available via <http://www.nrg.ee.lbl.gov/ns>, 1995.
- [19] E. Palmer. *Graphical Evolution: An Introduction to the Theory of Random Graphs*. John Wiley & Sons, 1985.
- [20] C. Papadopoulos, G. Parulkar, and G. Varghese. An error control scheme for large-scale multicast applications. In *Proceedings of IEEE INFOCOM’98*, San Francisco, USA, March 1998.
- [21] K. Sabnani, J.C. Lin, S. Paul, and S. Bhattacharyya. Reliable Multicast Transport Protocol(RMTP). *IEEE Journal on Selected Areas in Communication*, April 1997.
- [22] R. van Renesse. Masking the overhead of protocol layering. In *Proceedings of ACM SIGCOMM’96*, pages 96–104, August 1996.
- [23] R. van Renesse, Y. Minsky, and M. Hayden. A gossip-style failure detection service. In *Proceedings of Middleware’98*, September 1998.