Scalable, Low-Overhead Network Delay Estimation *

Volkan Ozdemir[†] *S. Muthukrishnan*[‡] *Injong Rhee*[†]

[†]Department of Computer Science
North Carolina State University
Raleigh, NC 27695-7534
{vozdemi,rhee}@csc.ncsu.edu

[‡]AT&T Labs-Research
180 Park Avenue
Florham Park, NJ 07932
muthu@research.att.com

Feb 1999

Abstract

Estimating the network delays between each pair of nodes in a multicast session is the key parameter in reliable multicast; it is used, among other things, in suppressing the implosion of request and repair packets, and in detecting congestion. Existing implementations use $O(n)$ multicasts with $O(n)$ message size each (total of $O(n^2)$ bits); here, n is the session size. We present a new protocol that requires $O(n)$ multicasts only with $O(1)$ message size each. We also present another protocol that estimates delays from each receiver to the sender without causing feedback implosion. Because of reduced message complexity and feedback implosion, these protocols enhance the scalability of reliable multicast. Furthermore, they do not require synchronized clocks, or any knowledge of network topology or the session size.

1 Introduction

We consider a fundamental technical problem in networks, namely, how to estimate round trip delay times between receivers in a multicast session in a scalable manner. Our main result is a procedure for solving this problem using fewer messages or bits than previously known methods. We integrate this solution with existing reliable multicast protocols – the combined protocols are substantially more scalable than the originals. Furthermore, we provide experimental evidence that our estimation methods are accurate. In what follows, we motivate and define our problem formally before presenting our results. Round trip delay time is a key parameter in unicast and multicast scenarios. In unicast communication such as in conventional TCP, it is used for timeouts at the source. In multicast communication, the estimated delays between all pairs of nodes are used to suppress feedback and repair packets. In recently proposed congestion control mechanisms [1, 2, 3], delay estimation from the source to each receiver in a multicast session is used to compute the possible transmission rate of the source.

Consider a multicast environment comprising a source and a large collection of receivers; they may be organized into a single scope, or multiple ones. The goal is to estimate round trip delay times between the source and the receivers, or between any pair of receivers. Following two assumptions are crucial.

*This work is supported in part by NSF CAREER ANI-9875651.

1. The clocks at receivers are not synchronized with each other, or with that of the source. In any wide area network including the current Internet it is difficult to guarantee synchronized clock and hence, this is a realistic assumption. [1]

2. The round trip delay times along paths are symmetric. This is standard in many network protocols involving delay estimation such as TCP[4], SRM[5], NTP[6], and SHARQFEC[7], to name a few.

Clearly the delay values are dynamic, changing with the traffic pattern and congestion in the network. In fact, these values may be significantly affected by the overhead of traffic generated by the protocol that performs the estimation by actively injecting packets into the network; hence, any such protocol must minimize this traffic overhead. Another motivation for reducing the traffic overhead is to increase the scalability of the estimation process, since it has to be run frequently to be up-to-date.

We adopt the standard practice in this area to measure the overhead of estimation procedure in terms of the number of messages, each unicast or multicast transmission being counted as one message.

Contribution Existing delay estimation protocols use $O(n)$ multicasts with message size $O(n)$ each, or $O(n^2)$ messages of size $O(1)$ each [5, 7]. Multicasting protocols, such as in SRM, must engage in frequently estimating the pairwise delay between receivers in a scope.

The currently best known estimation process involves each receiver in a scope to multicast a *session* message involving delay information about all the other receivers in the scope [7, 8]. The delay information consists of at least two time stamps. Let us do a quick calculation to estimate the bandwidth requirements for this session traffic. Say we have a scope with 500 members; the size of a session message is as large as $500 \times 2 \times 4 = 4$ KB, assuming a micro-second resolution on the time scale (i.e., 4 bytes for each time stamp). If we set the interval between two consecutive session message transmissions to be one second, each receiver will be subject to the total session traffic of $4 \times 500 = 2$ MB/s which is prohibitive. On the other hand, if we allow only 5 KB/s bandwidth for session traffic, then the time interval between session messages has to be adjusted to about 7 minutes, which does not faithfully represent changing delays between receivers. Thus, with the existing delay estimation protocol, SRM cannot be widely deployed for large-scale reliable multicast.

We present a protocol that for each receiver, estimates the delay from it to each of the others in its scope during a multicast transmission session. Our protocol uses $O(n)$ multicasts with $O(1)$ message size each; here, n is the size of a scope (or the session size if singly scoped). Our protocol thus has low overhead, and is hence more scalable than the existing ones. More precisely, our protocol uses at most $2n$ multicasts after an initial bootstrap phase. [2] Thus, with $n = 500$, it consumes approximately 10 KB/s session bandwidth (not counting the header size) if the session interval is set to one second; this is quite moderate compared to the 2MB/s estimate for the existing protocols. Also, our protocol does not require any knowledge of network topology and the identities of other members.

We also present a protocol that allows each receiver to estimate the round trip delay between the sender and itself without causing the receiver to send a message directly to the sender, hence avoiding implosion.

[1] In emerging systems, there is a proposal to interface with GIS so that the clocks at all the nodes in the Internet may be accurately synchronized; currently no such global system exists. In the future, if such systems are deployed, the problem of round trip delay estimation becomes trivial.

[2] The protocol can be optimized to incur only n messages if we allow one time stamp to be piggy-backed in the data packets sent by the source.

The protocol is applicable to hierarchically structured multicast protocols such as RMTP[9], TMTP [10], and SHARFEC[7].[3]

We incorporate our delay estimation protocols into the well-known SRM protocol and show performance enhancements through a simulation-based experimental study using the UCB/ISI/VINT network simulator (ns).

Organization In Section 3, we present our protocol for a single scoped multicasting environment. In Section 4, we show how to modify it for the hierarchically scoped scenarios. In Section 5, we describe our experimental setup and our results.

2 Related Work

Kermode [7] and Sharma *et al.* [8] propose to impose a hierarchy on SRM to solve the scalability problem induced, in part, by session traffic involved in estimating the pairwise network delay. The session members are divided into local scopes each of which contains a local representative. Local members within a scope send session messages to each other and conduct delay estimation among themselves while representatives perform their own delay estimation among each other. Delays between two members in different scopes are approximated via delays to their representatives. Under a small (local and global) scope comprising 20 to 40 members this technique is shown to reduce the bandwidth utilization of the delay estimation protocol.

Although hierarchical scoping helps increase the scalability of the delay estimation protocol in SRM, it still does not remove the $O(n^2)$ performance bound (hence, the limit on the scalability) where n can be the size of a scope. Thus, only small scopes can be accommodated. In addition, these protocols assume that scope representatives are on the multicast routing path from the sender. This assumption requires that scope configuration protocols dynamically defining scopes and electing representatives from each scope is aware of underlying network topology and routing mechanism.

In contrast, our delay estimation protocols (1) have a better performance bound, (2) can also be incorporated into many protocols (including scoped or non-scoped protocols) that require scalable, non-intrusive delay estimation, and (3) do not assume any synchronized clocks or knowledge of network topology.

3 Estimating network delays in a single scope

The problem we address in this section is to estimate the delay incurred by a transmission from any receiver (or the sender) to any other receiver or the sender, all in a single scope. The protocol consists of two phases: *setup* and *estimation* phase.

We describe each phase in detail now.

Setup phase. In this phase, each receiver R determines the network delay $d(S, R)$ from the sender S to itself. In order to do this, each receiver sends its current time in message to the sender. The sender merely

[3] Basu and Golestani[11], and we presented a similar hierarchical protocol at the Reliable Multicast Research Group meeting in Arlington, VA Dec 3-4, 1998. Only the work on the hierarchical protocol was performed both concurrently and independently by the two research groups.

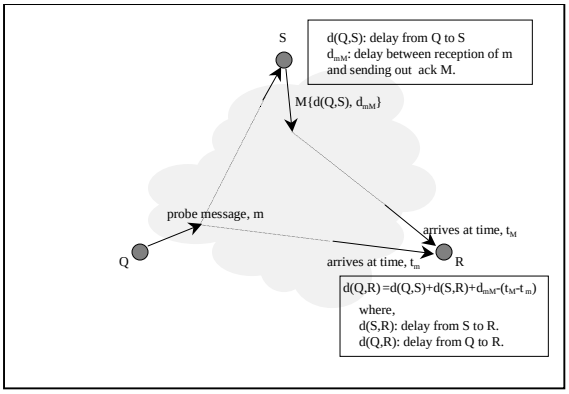

Figure 1: The delay estimation phase

sends the same message back to each of the receivers. By comparing the time this message is received with the time stored in the message, each receiver can compute the delay $d(S, R)$. As stated, this phase uses at most $2n$ messages each of $O(1)$ size (assuming the granularity of time scale to be micro-seconds, the size can be as large as four bytes). This phase can be simplified by letting the sender multicast a message containing the concatenation of all their message contents back to the receivers. This requires n messages of size $O(1)$, and one message of $O(n)$.

Estimation phase In this phase, each receiver R determines the delay from every other receiver Q to itself; here, $d(S, R)$ is used crucially.

Figure 1 illustrates this part of the protocol. *Fix* a receiver Q. We focus on the problem of each receiver R estimating the delay of transmissions from Q. Receiver Q multicasts a *probe* message m containing $d(A, S)$ and the (local) time it sends m. Say the time this message is received by a receiver R is t_m. The sender, upon receiving m, multicasts a message M comprising d_{mM}, the delay between receiving m and sending out M. Say M is received by R at time t_M. The following holds.

Lemma 3.1 $d(Q, R) = d(Q, S) + d(S, R) + d_{mM} - (t_M - tm)$.

Proof. Let t be the time (in R's clock) that Q sent m. We can estimate the time t in two different ways. Since m was received at R at time t_m, $t = t_m - d(Q, R)$. Also since M was received at R at time t_M, $t = t_M - d(S, R) - d_{mM} - d(Q, S)$. Both estimates of t must be identical, hence $t_m - d(Q, R) = t_M - d(S, R) - d_{mM} - d(Q, S)$ from which the lemma follows. ∎

Thus every receiver R can compute $d(Q, R)$ for a fixed Q. This takes 2 multicast messages each of size $O(1)$. Now, we repeat this process with each receiver R playing the role of Q; at the end of this, every receiver R has $d(R, Q)$ for each Q. The whole process requires $2n$ multicast messages each of size $O(1)$. We make two remarks.

Remark 1. One can optimize the number of messages to be n by allowing the sender to piggyback message M to its data packets. Further optimization can also be achieved by allowing the sender to collect probe

messages (m's) from *all* the receivers, and to multicast a cumulative acknowledgment M to all the receivers that contains d_{mM} for all m's. This requires n probe messages each of size $O(1)$ and a single multicast message from the sender of size $O(n)$.

Remark 2. The overall protocol is to run the setup and estimation phases alternately. By slightly modifying the estimation protocol, we can ensure that following a *single run* of the setup phase, repeatedly running the estimation protocol itself suffices. The receiver Q now sends the message m comprising (a) the time the sender sent the last packet received by Q from the sender (this information can be obtained from the packet) and (b) the delay between when the last packet from the sender was received by Q and the current time when Q sends message m. The sender can determine $d(Q, S)$ from this information. Sender now broadcasts $d(Q, S)$ and d_{mM} as before, and the rest follows easily. The message complexity remains unchanged.

The pseudo-code of this protocol appears below.

Receiver i's protocol

At every period T_{probe}:
probe_sequence ++;
$t_s \leftarrow$ the time stamp in the last message from the sender;
delay $\leftarrow$ the delay from the reception of the last message from the sender
multicast a probe P_i(*probe_sequence*,t_s,*delay*)

On receiving a probe P_i(*s*,t_s,*delay)*: //received a probe message with sequence number s.
Probe$[i]$.*time* $\leftarrow$ *gettimeofday*();
Probe$[i]$.*seq* $\leftarrow$ *s*;
if (*Ack*$[i]$.*seq* = *s*) // if received the corresponding ack from the sender
$D[i, j] = (D[i, S] + D[S, j] -$ *Ack*$[i]$.*delay) - (Probe*$[i]$.*time - Ack*$[i]$.*time)*;

On receiving an ack *ACK*(*i,s,delay,*$d_{i,S}$) from the sender S,
Ack$[i]$.*time* $\leftarrow$ *gettimeofday*();
Ack$[i]$.*seq* $\leftarrow$ *s*;
Ack$[i]$.*delay* $\leftarrow$ *delay*;
$D[i, S] \leftarrow d_{i,S}$;
if (*Probe*$[i]$.*seq* = *s*) //if received the corresponding probe from i
$D[i, j] \leftarrow (D[i, S] + D[S, j] -$ *Ack*$[i]$.*delay) - (Probe*$[i]$.*time - Ack*$[i]$.*time)*;

Sender's protocol

On receiving a probe $P_i(s, t_s, delay)$:
$t_{now} \leftarrow$ *gettimeofday*();
$D[i, S] \leftarrow t_{now} - t_s -$ *delay*;
delay $\leftarrow$ the delay from the reception of the probe message;
multicast *ACK(i,s,delay,*$D[i, S]$);

4 Estimating network delay in a hierarchical scope

Our discussion on network delay estimation in Section 3 focuses only on how to estimate delays between receivers within a single scope. The protocol there relies on the fact that the feedback from the receivers

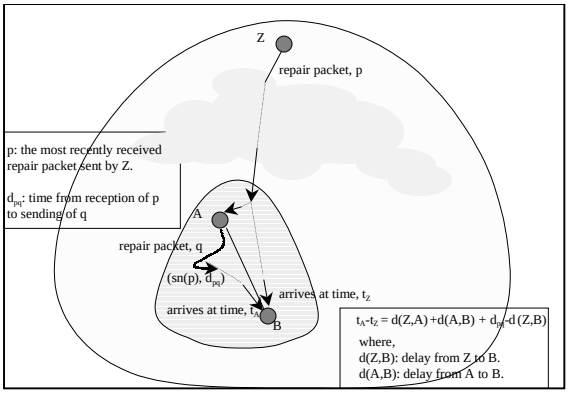

Figure 2: Hierarchical delay estimation

is multicasted to the entire scope. In presence of hierarchical scopes, the entire multicast group is broken into hierarchically nested scopes and designates a receiver within each scope as the *zone closest receivers* (ZCRs) [7] or *designated receivers* (DRs) [9]. In these protocols, typically receivers send feedback only to the ZCRs of their own scopes (i.e., their parents). Feedback transmitted directly to the sender or receivers outside their scopes is avoided to increase scalability.

Hierarchically scoped protocols, such as SHARQFEC [7], deal with this difficulty by assuming that ZCRs are at the multicast routes from the sender, and successively adding delays between a parent ZCR and its child ZCR. However, this assumption holds only when receivers has the knowledge of network topology.

In this section, we present a protocol that estimates the network delay from a ZCR Z (including the sender) to its receivers in its nesting scopes without any topological assumption or synchronized clocks. No feedback from the descendent receivers other than from the immediate children of Z is necessary. As before, we assume that network delays are symmetric.

The protocol applies the principle discussed in Section 3 recursively in the hierarchical setting. Figure 2 illustrates the intuition behind the protocol. The objective is to estimate delays from Z to a receiver B ($d(Z, B)$). Suppose that receiver A is the ZCR of B, and A is a descendent of Z. Note that Z can multicast repairs (or data if the sender) to both A and B. The delay between A and B ($d(A, B)$) can be computed as described earlier (i.e., B includes in the feedback the time stamp in the last message received from A).

Suppose that receiver B knows the delay from Z to A ($d(Z, A)$) (we will discuss how to estimate this delay lator). Then $d(Z, B)$ can be estimated by B as follows. First Z multicasts a repair packet p to its base group to which both A and B must belong. Let t_Z be the time that B receives this packet according to B's clock. When A receives p, it multicasts packet q containing the sequence number of the last packet it received from Z and the delay from the reception of that packet to the sending of q which is denoted d_{pq}. Let t_A be the time that B receives packet iq from A. We have the following: $t_A - T_Z = d(Z, A) + d(A, B) + d_{pq} - d(Z, B)$. It follows that, $d(Z, B) = d(Z, A) + d(A, B) + d_{pq} - (t_A - T_Z)$.

Now the remaining issue is ensure that receiver B knows the time delay between Z and A. Note that all the ancestor ZCRs of B (including A) can estimate the delay from Z by applying the same argument recursively while the delay between Z and its immediate child receivers (which is a base case in the recursion) can be estimated in the usual way. Thus, A knows the delays from all of its ancestor ZCRs of its nesting

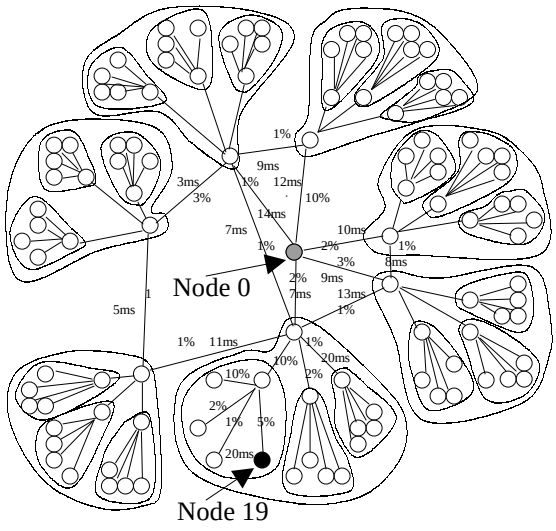

Figure 3: Network simulation topology with 113 nodes

scopes. This information can be piggy-backed on the repair packets it multicasts to B. This information would occupy no more than $O(h)$ space where h is the height of the tree of scopes. This does not add much overhead.

5 Simulation

5.1 Simulation setup

We implement our delay estimation protocol in UCB/LBNL/VINT, and study the performance of SRM with this new protocol. Our overall experimental setup is very similar to the one in [7].

Topology. Our simulation experiments were run using variants of the hybrid mesh tree topology used in [7]. The two topologies used in our experiments are shown in Figure 3. their configuration is identical to the ones in [7] except for the loss rates assigned to each link. The sender at node 0 feeds data to a three level hierarchy of 112 receivers arranged as a mesh of 7 receivers each of which feeds balanced trees. Each of seven trees in the topology is an exact copy of each other, and three subtrees within each tree are also a copy of each other. The links connecting the source to the top 7 nodes in each tree are 45Mbits/s with all other links set to 10 Mbits/s. The latency between any two receivers located within each tree was set to 20 ms for each link while the latencies used for the backbone links are shown in the figures. The loss rates for the links are varied over different parts of the networks.

The loss model. To simulate a realistic loss behavior, we conducted transmission experiments over a transpacific link every 45 minutes between Oct. 10 and Oct. 13, 1998, and recorded all the packets being

Time(sec)	Our Protocol	SRM
1	9056	753792
2	25824	1710528
3	24448	1652544
4	23840	1594560
5	24576	1623552
6	24960	1710528
7	24160	1609056
8	23648	1638048

Table 1: Bandwidth (bits/sec) used for delay estimation protocols measured at each second of simulation

received and lost. We gathered over 100 traces each 15 minutes long, and extracted the profile information of each trace which comprises the loss characteristics of every non-overlapping 300 ms segment. The loss characteristics include the number of instances of loss bursts of lengths from 1 to over 200. For each link in the networks shown in Figure 3, we find a trace that undergoes the same average loss rate as that of the link, and use the trace to pick packets to drop during each 300 ms period. The loss rates for links are shown in Figure 3. The loss rates that receivers experience can be obtained by compounding the loss rates on the links from the sender to the receivers. In our topologies, they vary from 1% to 27.5%. Every packet passing through a link — data, repair, request, and session — is subject to the same loss rate indicated on that link.

Transmission. Each simulation experiment starts the session at time 1 second, at which time nodes begin sending session messages, and after the initial bootstrap phase of 6 seconds, node 0 starts sending traffic at a constant bit rate of 800 Kbits/s. Each data packet is 1024 bytes. The sender stops transmitting data packets at time 16 seconds. Note that at this transmission rate, each receiver will get approximately 10 packets over a 100 ms period.

5.2 Simulation result

We implemented our delay estimation protocol and compared its overhead and performance with that of SRM's session protocol. We use the topology shown in Figure 3 for simulation experiments. Table 1 compares the bandwidth overhead of our protocol with that of the SRM session protocol. Here the time interval between two consecutive transmissions of delay estimation probe (or session message) by the same receiver is set to one second. Our protocol uses only about 20 Kbits/s while the SRM session protocol uses over 1.6 Mbits/s. This is because the size of session messages in SRM is proportional to the number of receivers in the session, and each receiver sends at least one message per second. In contrast, our protocol uses only a constant size message.

To see the the accuracy of our delay estimation, we measure the ratio of actual network delays experienced by each packet to our estimated delays. For this purpose, we randomly picked node 19 in the topology and measured the delays that node 19 experiences and compared its estimated delays. The actual delays can be computed using time stamps embedded in each packet. Since the simulation uses synchronized clocks actual delays can be easily computed by taking a weighted average of each sample. Figure 4 shows that our estimated delays are well within a few percents of the actual delays.

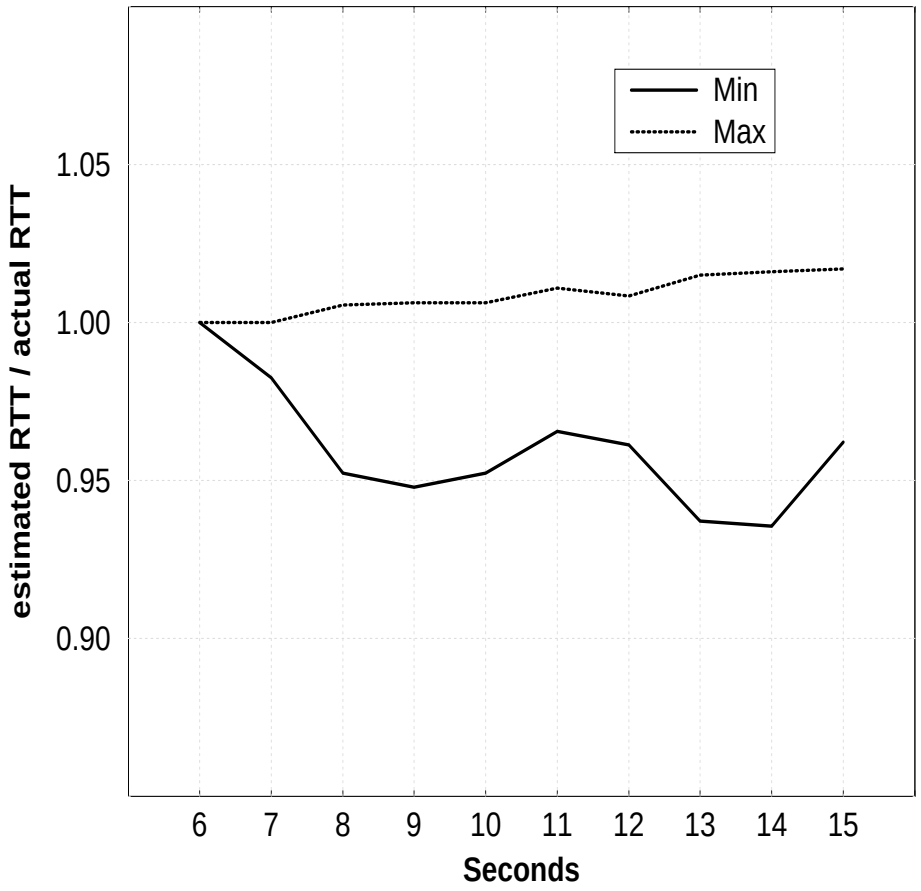

Figure 4: Ratio of estimated RTTs over actual RTTs sampled over the running time period of SRM

We also incorporated our protocol into SRM: we modified SRM to use our delay estimation to set its suppression timers. Figures 5 and 6 compare the average number of request and repair packets per receiver for the original SRM and that for the modified SRM. The graphs shows that using our scheme does not change the request and repair message traffic by a significant amount. This indicates that our delay estimation protocol can be used for suppressing request and repair as effectively as that in the original SRM that requires $O(n^2)$ message bit complexity.

6 Conclusion

We study the scalability of reliable multicast protocols. Some of the fundamental known techniques for devising scalable reliable multicast protocols include receiver-centric repair [5], scoping [7], suppressing

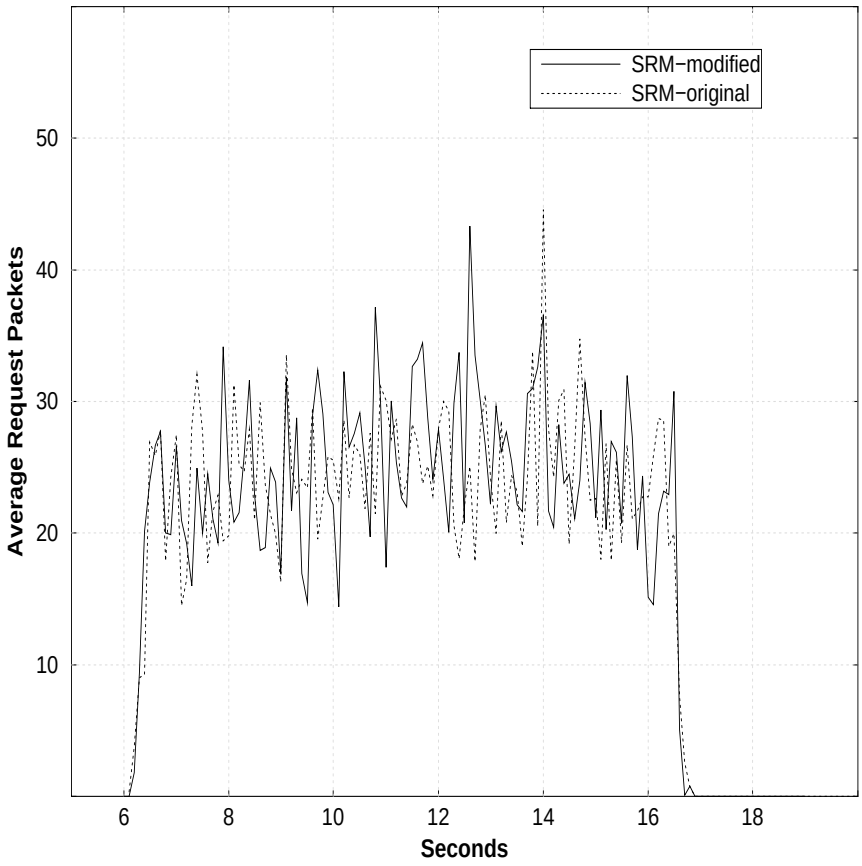

Figure 5: The impact of our protocol on the request suppression

unwanted traffic by using timers [5], use of FEC packets [12, 13, 7, 14, 15] etc. In this paper, we offer one additional tool, namely, a low-overhead protocol for estimating network delay between the receivers. We employ this tool on existing protocols and present simulation results that show the combined protocols to be substantially more scalable.

Our protocol runs on the assumption that delays between two processes are symmetric. This assumption may not hold in some situations such as sattelite or mobile wireless communications, and some wired lines. Our results on estimating oneway delay times in asymetric settings will appear in [16]

References

[1] M. Handley and S. Floyd. Strawman congestion control specification, the reliable multicast research

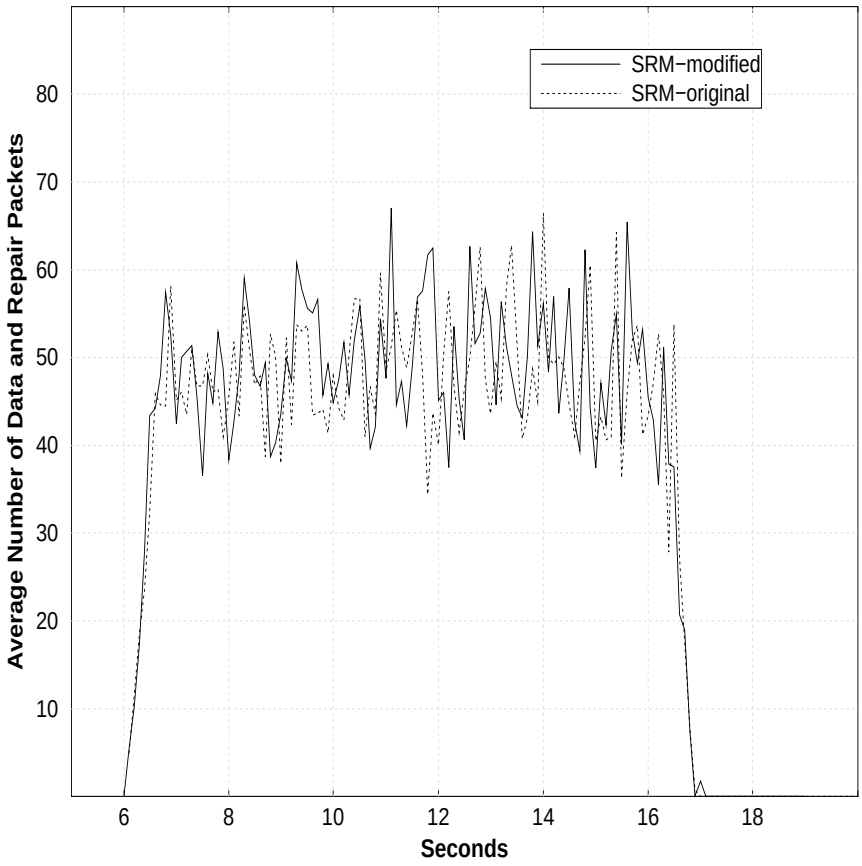

Figure 6: The impact of our protocol on the repair suppression

group meeting in Arlington, VA. Dec 1998.

[2] I. Rhee, N. Balaguru, and G. Rouskas. MTCP: Scalable TCP-like congestion control for reliable multicast. In *Proceedings of the INFOCOM*, New York, NY, March 99.

[3] B. Whetton and J. Conlan. A rate based congestion control scheme for reliable multicast. Technical report, GlobalCast Communication, Oct 1998.

[4] V. Jacobson. Congestion avoidance and control. *ACM Communications Review*, pages 314–329, August 1988.

[5] S. Floyd, V. Jacobson, S. McCanne, C. G. Liu, and L. Zhang. A reliable multicast framework for light-weight sessions and application level framing. In *Proceedings of the ACM SIGCOMM Conference*, pages 342–356, October 1995.

[6] D. Mills. Network time protocol(v3). April 1992.

[7] R. G. Kermode. Scoped hybrid automatic repeat request with forward error correction (SHARQFEC). In *Proceedings of the ACM SIGCOMM Conference*, pages 278–289, October 1998.

[8] P. Sharma, D. Estrin, S. Floyd, and L. Zhang. Scalable session messages in SRM using self-configuration. Technical report, USC, July 1998.

[9] S. Paul, K. K. Sabnani, J. C. Lin, and S. Bhattacharyya. Reliable multicast transport protocol (RMTP). In *Proceedings of the IEEE INFOCOM*, San Francisco, CA, March 1996.

[10] R. Yavatkar, J. Griffioen, and M. Sudan. A reliable dissemination protocol for interactive collaborative applications. In *Proceedings of ACM Multimedia*, 1996.

[11] A. Basu and J. Golestani. Estimation of round trip times in multicast comnunication. Technical report, Bell Labs, Dec 1998.

[12] J. Nonnenmacher, E. Biersack, and D. Towsley. Parity-based loss recovery for reliable multicast transmission. In *Proceedings of ACM SIGCOMM*, September 1997.

[13] J. Gemmell. Scalable reliable multicast using erasure-correcting re-sends. Technical Report MSR-TR-97-20, Microsoft Research, June 1997.

[14] Dan Rubenstein, Jim Kurose, and Don Towsley. Real-time reliable multicast using proactive forward error correction. In *NOSSDAV 98 (to appear)*.

[15] D. Rubenstein, S. Kasera, J. Kurose, and D. Towsley. Improving reliable multicast using active parity encoding services(apes). In *To appear in IEEE Infocom '99*.

[16] S. Muthukrishnan and I. Rhee. Estimating one-way delys in asymmetric multicast networks, manuscript, May 1999.