Impact of Network Dynamics on End-to-End Protocols: Case Studies in TCP and Reliable Multicast[1]

Kannan Varadhan *Deborah Estrin*

USC/Information Sciences Institute
Marina Del Rey, CA 90292.
+1 310 822 1511
+1 310 823 6714 (Fax)
{kannan,estrin}@isi.edu

Sally Floyd

M/S 50B-2239B, Lawrence Berkeley Laboratory
One Cyclotron Road, Berkeley, CA 94720.
+1 510 486 7518
+1 510 486-6363 (Fax)
floyd@ee.lbl.gov

April 3, 1998

.

Abstract

End-to-end protocols measure network characteristics and react based on their estimates of network performance. Network dynamics can alter the topology significantly, and thereby affect protocol operation. Topology changes may result in routing pathologies (such as route loops, packet interleaving), changes to the end-to-end path characteristics, network partition *etc*, that then impact the performance of end-to-end protocols. This paper presents methodologies to evaluate an end-to-end protocol in the presence of network dynamics using a simulator. A number of models of network dynamics are introduced. In the paper, we evaluate two end-to-end protocols over dynamic topologies and study the adaptivity of these protocols to topology changes. The first part of the paper illustrates the effect of packet interleaving on the performance of TCP. The second part of the paper is a systematic evaluation of the adaptive timer mechanisms in Scalable Reliable Multicast (SRM). The timer mechanisms are evaluated under simple topology changes, as well as under network partition conditions. The paper concludes by posing a number of open research questions about the behaviour of different reliable multicast mechanisms when operating over dynamic topologies.

Keywords: Protocol Design, Protocol Evaluation, TCP, Reliable Multicast, SRM, Network Dynamics, Topology Changes.

Areas of Interest: Protocol Design and Analysis, Internetworking, Multicast/Broadcast Algorithms

[1]The work of K. Varadhan and D. Estrin was supported by the Defense Advanced Research Projects Agency (DARPA) under contract number ABT63-96-C-0054. The work of S. Floyd was supported by the Director, Office of Energy Research, Scientific Computing Staff, of the U.S. Department of Energy under Contract No. DE-AC03-76SF00098, and by ARPA grant DABT63-96-C-0105. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the DARPA or the U.S. Department of Energy.

[1]This technical report is USC-CS-TR 98-672, and is an extended version of [21].

1 Introduction and Related Work

A number of transport and application level protocols measure and react to network characteristics, such as the end-to-end delay, available bandwidth, *rtt*, congestion, *etc*. A TCP [17] source adjusts its data rate based on its estimate of the available bandwidth and congestion; Receiver-Driven Layered Multicast (RLM) [14] requires a group member to estimate the level of congestion, and decide the amount of data that it can receive at any given time; in Scalable Reliable Multicast (SRM) [4], every group member estimates the *rtt* to all other group members, for use in its error recovery mechanisms. Common to all of these protocols is that changes to the topology can significantly change the actual network characteristics, and therefore impact the performance of the protocol, and possibly adversely affect the network.

Paxson [16], through end-to-end measurements of routing stability, tells us that in general, the stability of a particular end-to-end path is fairly high (around 60%); *i.e.*, if, at a given instant in time, an end-to-end transport session uses a particular route from end-to-end, then there is a 60% likelihood that the session will use the same route at a later instant in time. However, he notes further that, there is extensive variability in the measure from site to site (variability of 50%–60%), and therefore, while one end-to-end path may remain fairly stable, another might be less so. Govindan *et al.*[5] arrive at a similar conclusion, based on their analysis of inter-domain routing protocol traces gathered at various points in the Internet backbone over 18 month period. More recently, Labovitz *et al.*[9], through measurements of inter-domain route updates at various network access points, suggests that there are a far greater number of route updates than is to be expected in a network, the size of the Internet.

Paxson [16] also points out a number of routing and network pathologies that adversely affect TCP performance. By design, multi-party protocols and applications support a greater number of participants; they are used to communicate simultaneously with participants connected by diverse network paths. Therefore, it is reasonable to expect that routing and network pathologies can have even greater impact on the performance of multi-party protocols.

Transport protocols are designed to be adaptive to varying network conditions, but their evaluation is often conducted after their deployment on an operational network. We need to understand the full range of protocol behaviour before protocol deployment, in order to be sure that it meets the desired design characteristics. In addition, tradeoffs must be evaluated with respect to possibly conflicting goals in the design process. A common example of this is the tension between the goals of robustness and efficiency in adaptive protocols. We propose to enhance the capabilities of the protocol designer to evaluate protocols in the presence of dynamic topologies. This approach permits the designer to more comprehensively understand protocol behaviour, and make the appropriate design choices.

Shankar *et al.*[19] in analyzing different routing protocols, describe the impact of network dynamics on the throughput and delay of a simple window based transport protocol. Individual protocol designers sometimes suggest how their protocol should perform in the presence of topology changes [4]; however, we know of no systematic study of the performance and evaluation of end-to-end protocols over dynamic topologies.

In this paper, we look at some of the issues involved in studying transport protocols over dynamic topologies. We concentrate on the study of two protocols: TCP and SRM. TCP [17, 3] is a reliable transport protocol for unicast connections. TCP has been extensively studied, both via simulations and on operational networks. However, its detailed performance analysis in the presence of dynamic topologies has never been reported. We develop our mechanisms using TCP as an example (Section 2); this also helped us validate our methodology.

A number of reliable multicast protocols have been proposed in the literature recently, such as Lin *et al.*[10], Yavatkar *et al.*[23], Holbrook *et al.*[8], Floyd *et al.*[4]. In our paper, we analyze the timer mechanisms in one reliable multicast protocol: SRM (Section 3). We present results of our preliminary work with SRM when it is running over dynamic networks.

We close with a discussion of a number of promising research issues related to the impact of network dynamics on reliable multicast mechanisms (Section 4).

2 Impact of Route Dynamics on TCP

In this section, we will use TCP as an example to demonstrate our methodology of studying a protocol over dynamic topologies. TCP [17] is a unicast transport protocol that guarantees ordered reliable delivery of data. There are a number of studies of TCP in the literature. Fall *et al.*[3], in particular, has compared different flavours of TCP over stable topologies. Our work in this section is an extension to this earlier work, studying the behaviour of TCP over dynamic topologies. We separate the notion of a dynamic topology from the routing protocol that is used to repair the topology.

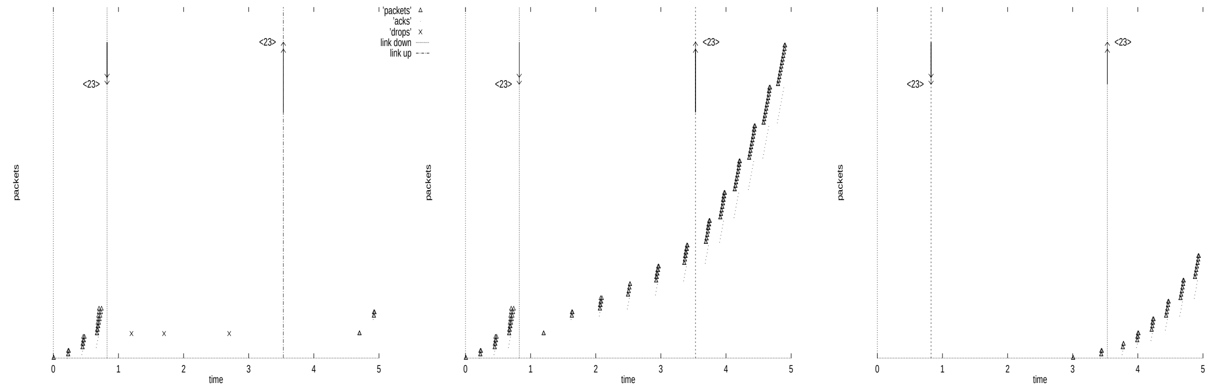
In any topology, a network is dynamic if one or more nodes or links in the topology can periodically fail and (possibly) recover. A number of different models of network dynamics can be used to specify the durations when a particular node or link will be down or up. A routing protocol must then be used to compute the end-to-end connectivity after any topology change.

As with the models of dynamics, different types of routing protocols can be used. In a typical simulation that has no network dynamics, the user can compute the routes in the topology off-line, and then start their simulation. We call this one-time off-line route computation strategy, “Static” routing. A similar off-line route computation strategy can be used when the network is dynamic. In this strategy, routes are recomputed over the topology after any topology change. We call such an off-line route computation strategy, “Session” routing. A more realistic option is to simulate “Dynamic” routing.

Figure 1 shows a TCP session running over a dynamic topology, with three different kinds of unicast routing strategies. The experiments in this figure use the topology of Figure 2. The figures show Link $\langle 2, 3\rangle$ fail at $t \approx 0.8s$, and recover at $t \approx 3.5s$. The plot shows the failure and recovery events as vertical lines at the appropriate times; the arrows at the top of the graph shows the type of event, the label adjacent to the arrows indicate the relevant link. An alternate path is available to route around the failed link. We notice that the throughput and response of TCP is different depending on the type of routing strategy used in the simulation.

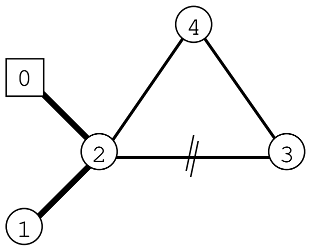
Each type of routing strategy has specific consequences for the transport protocol that is being simulated. Static routing, if used over a dynamic topology, can lead to temporary partitions while a node or link is down. This explains why there is no packet throughput seen in Figure 1(a) when Link $\langle 2, 3\rangle$ is down Session routing, on the other hand, will always ensure end-to-end connectivity, as long as the topology is connected. However, results from a simulation with Session routing at the network layer, can be incorrect. For example, throughput measured by a protocol simulation that uses Session routing to repair network dynamics will often be higher than is normally possible. Figure 1(b) shows that a greater number of packets are successfully transmitted than occurs in the adjacent traces.

While Dynamic routing is more realistic, it entails overhead, and can introduce its own artifacts. For example, a protocol designer must take into consideration the fact that, at startup time ($t = 0s$), no node in the topology has routes to other nodes, except possibly to its directly connected neighbours. Dynamic routing takes some time



(a) Static routing (b) Session routing (c) Dynamic routing

Figure 1: Comparison of different routing protocol options in a simulation



Brief notes on the topology:
— Source of interest is at Node 0
— Link $\langle 2, 3 \rangle$ is dynamic in selected experiments

Figure 2: Topology used for first set of TCP experiments

to become quiescent after some number of routing protocol messages are exchanged by all of the nodes in the topology. From Figure 1(c), we can infer that the TCP session looses all of the initial packets. This is because the session starts at $t = 0s$ in our simulation, and at this time the route computation has not yet become quiescent. Since the retransmit timeout is very high when a session's data packets are lost very early in the connection history (and this timeout value is set at $3s$), we see the first successful packet transmission at $t = 3s$ via the alternate path.

For the rest of the experiments in this paper, we use a Dynamic routing protocol. We have implemented a simple Distributed Bellman-Ford routing protocol in `ns`[13]. Our choice of parameters for the routing protocol are motivated by our desire to study the effect of topology changes on end-to-end protocol. One well-understood effect of topology change on these protocols is a transient loss of connectivity. The duration of this connectivity is a function of the routing protocol and the topology. While this area of research is quite interesting and useful when trying to understand protocol behaviour, our focus is on other effects that arise from topology changes, such as intra-network behaviours that affect protocol operation (for example, packet interleaving effects), or variations in the path characteristics (for example, sudden variations in the *rtt*). Therefore, we choose our parameters to ensure rapid routing protocol convergence without interfering with the simulation of the end-to-end protocols. In our simulations, the route update interval is $2s$. In addition, updates are triggered whenever the incident topology at a node changes state, or when that node receives an update from a neighbour that causes it to recompute new routes.

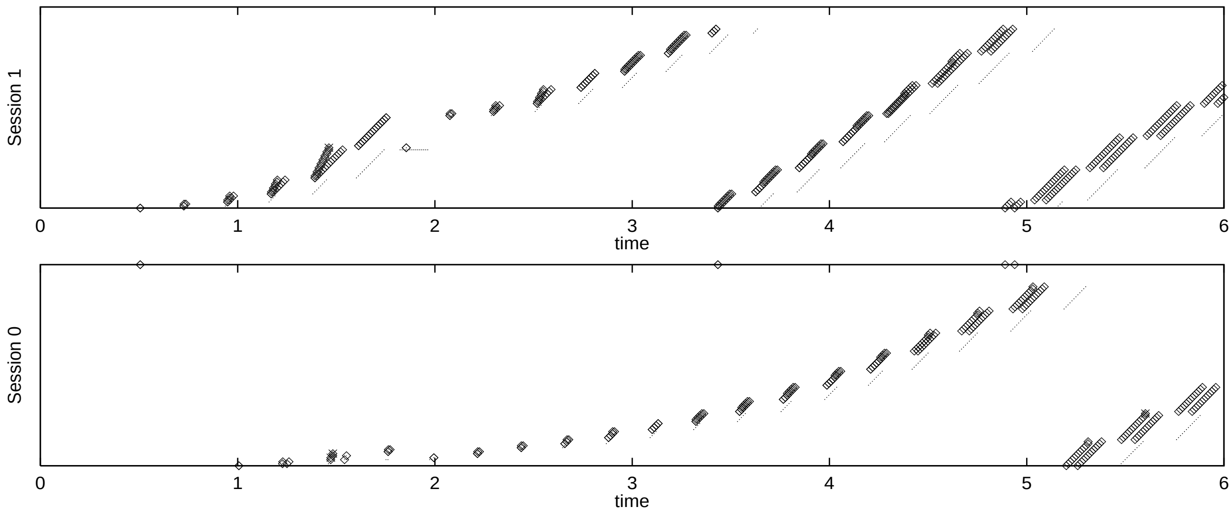


Figure 3: TCP Tahoe throughput over stable topologies

We use the topology shown in Figure 2 for the first set of our TCP simulations. In this topology, Links $\langle 0, 2\rangle$ and $\langle 1, 2\rangle$ have bandwidth $8Mb$, and propagation delay $5ms$. All the other links have bandwidth $800Kb$, delay $100ms$. Link $\langle 2, 3\rangle$ has a queue limit of 8 packets. This link is also dynamic, *i.e.*, it periodically fails and recovers. The link will be down (and up) for a period of time drawn from an exponential distribution with a mean of $1s$ (and $10s$). We simulate two FTP sessions, one from Node 1 to Node 3, starting at time $t = 0.5s$, and the other from Node 0 to Node 3, starting at time $t = 1.0s$. The window size used by the latter session is 100 packets; Figure 5 presents the time *vs.* sequence number plots of the session from Node 0 to Node 3, when it uses three different flavours of TCP: Tahoe TCP, Reno TCP, and TCP SACK.

The simulation configuration above (topology + unicast routing) differs from [3] in two significant ways: (1) their topology has no alternate paths between the sources and sink though Node 4, and (2) they use Static routing to precompute the routes in the topology prior to simulation execution. However, we can verify that neither of these changes affects protocol behaviour when the topology is stable. Figure 3 shows the throughput of the two FTP sessions using Tahoe TCP in stable topologies.[2]

In this figure (and all other figures in this section), we plot the sequence numbers modulo 90. We trace the sequence numbers of packets traversing Link $\langle 2, 3\rangle$. Each packet is plotted as two points: the time and sequence number of the packet when it is placed on the queue of the link that is being traced, and the time and sequence of the packet when it is removed from the queue. The spacing between the two points is a measure of the queuing delay experienced by a packet on that particular link. The sequence number of the acks for each packet are also shown as tiny dots. These acks are seen approximately one *rtt* after the packet was first sent by the source.

The upper plot in Figure 3 shows the throughput of Session 1, the FTP session from Node 1 to Node 3; the lower plot shows the throughput of Session 0, the other FTP session from Node 0 to Node 3. Each session experiences congestion and consequent packet drops at $t \approx 1.5s$. Following the packet drop, each session adjusts its window, and resumes transmission. This behaviour is identical to that described by Fall *et al.*[3].

Our focus of research is on protocol behaviour in the context of topology changes. We will therefore consider the periodic failure and recovery of Link $\langle 2, 3\rangle$; specifically, the link will fail at $t \approx 0.8s$ and recover at $t \approx 3.5s$

[2]This corresponds to the simulation of tahoe3, over the net0 topology, from the test-suite-routed.tcl test suite in the `ns`-2 distribution.

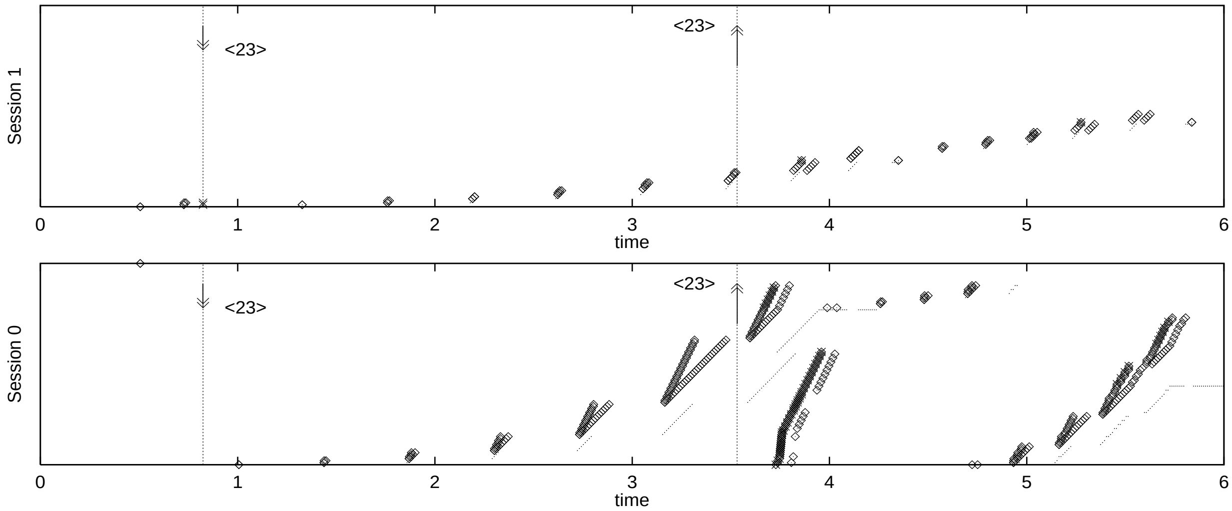


Figure 4: TCP Tahoe throughput over dynamic topologies

respectively. These events are shown as vertical lines at appropriate points in time. The alternate path through Node 4 is available when Link $\langle 2, 3\rangle$ fails. Our plots will show the trace of packets through Link $\langle 2, 4\rangle$ when Link $\langle 2, 3\rangle$ is down.

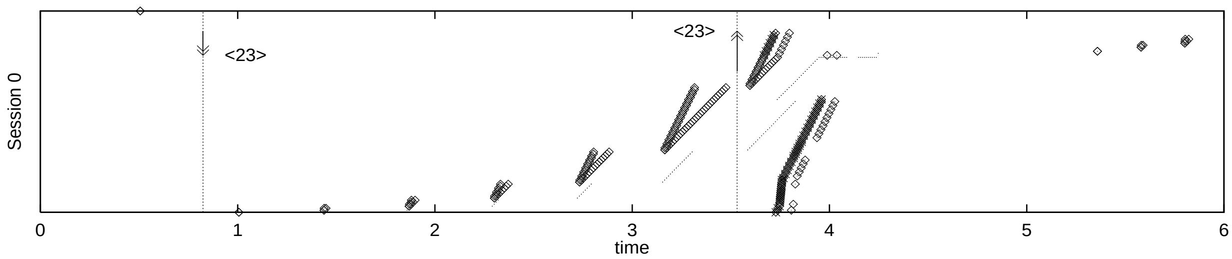
Unlike in the stable case, we see from Figure 4 that each session experiences different types of network behaviours.[3] Session 1 from Node 1 to Node 3 experiences packet loss due to link failure early in its connection phase (Figure 4). In fact, Session 1 in each of the variants of TCP experienced this same type of loss, and exhibited the same response to this loss. Hence, we do not plot the throughput of this session in our subseqent figures. Session 0 from Node 0 to Node 3, on the other hand, experiences the arrival of interleaved acks and significant packet drops following link recovery. We now discuss these two effects exhibited by this session from Node 0 to Node 3 (Session 0 in Figure 4, and Figure 5): the obvious packet drops and the interleaving of acks seen at about the same time.

We first make a couple of observations about the packet drops occuring at $t \approx 3.5s$ in each of the sessions in Figure 5. In this session, all of the packet drops occur in a single window of packets. These drops occur because of routing and topology changes. Hence, *routing changes can lead to multiple packet drops*. Moreover, these packet drops occur just after the recovery of Link $\langle 2,3\rangle$. This was contrary to our expectation that packet drops occur due to link failure, but never due to link recovery. Therefore, *packet drops may occur due a link recovery*.

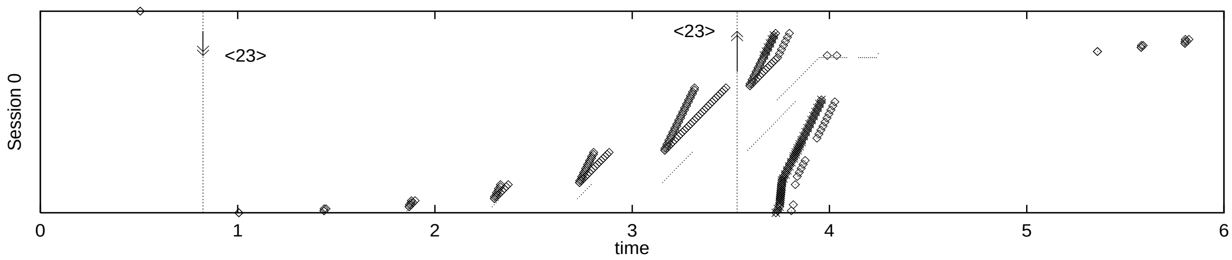
One of the reasons for this packet drop is the arrival of interleaved acks. These acks are in transit across both the original longer path, and the newer shorter path, at the instant of link recovery. In general, there are several possible consequences of such packet or ack interleaving.

One consequence from interleaved acks/packets is that the sender can get three *dup-acks*, and assume that a packet has been lost, even though the packet is not lost. This forces all variants of TCP to do a fast retransmit of the "lost" packet. TCP Reno, in particular, is susceptible into going into a retransmit timeout needlessly; TCP Reno is defined by this behaviour in response to multiple packet drops.

[3]This corresponds to a simulation of tahoe3 over the topology net0-DV, with link failures at $t = 0.8245s$ and $t = 3.5315s$ from test-suite-routed.tcl in the `ns`-2 distribution. The results of Figure 5 correspond to the reno3 and sack3 simulations in test-suite-routed.tcl and test-suite-sack.tcl, in the `ns`-2 distribution.



(a) Reno TCP



(b) SACK TCP

The sessions all create transient congestion after Link ⟨2,3⟩ recovers*; we can see the difference in the recovery mechanisms of the three sessions after the topology change event:*
- Reno TCP (Figure (a)), after a fast retransmit, waits for its RTO timer to fire, and then initiates slow start.
- SACK TCP (Figure (b)), appears to behave identically as Reno does, but has a slightly better throughput than Reno TCP.
- Tahoe TCP (Session 0 in Figure 4) initiates slow start immediately after a fast retransmit, and hence has the highest performance of the three.

Figure 5: Effect of interleaving acks on Reno and SACK TCPs

Another possible consequence of interleaved acks is that the sender suddenly sees a big jump in the sequence number acked, and therefore sends a large burst of packets back to back. This can then result in actual packet drops as opposed to simply perceived packet drops. We saw this earlier in Figure 5, when packets from three types of TCP were dropped.

In a different simulation of TCP SACK using the same topology[4] of Figure 2[5], we see both of these consequences (1. an unnecessary retransmit of out-of-order packets, and 2. a real packet drop), exhibited by a TCP SACK source (Session 0 in Figure 6). In the figure, the receiver, Node 3, receives a few packets out of order. These are six packets transmitted across Link $\langle 2, 4\rangle$ just before Link ⟨2,3⟩ recovers at $t \approx 3.5s$; they are in transit across the alternate longer delay path. Subsequent packets in the same window arrive at the destination earlier using the shorter Link $\langle 2, 3\rangle$. (1) The destination continually acks for the interleaved packets still in transit across Links ⟨2,4,3⟩; after receiving three dup-acks, the sender retransmits the six packets at $t \approx 3.8s$ even though they are not actually lost. (2) Subsequently, based on the jump in the sequence number acked by the receiver, the source sends a burst of packets (at time $t \approx 4.0s$) that then results in the actual packet drops. In contrast, Session 1 in

[4]The topology differs from the earlier one in that Link $\langle 0, 2\rangle$ is a $10Mb$ bandwidth, $3ms$ delay link, Link $\langle 1, 2\rangle$ is a $10Mb$ bandwidth, $3ms$ delay link; the other links have $1.5Mb$ bandwidth, $100ms$ delay. Link $\langle 2, 3\rangle$ has a queue limit of 23 packets.

[5]This simulation corresponds to a simulation of sack5 over the topology net1-DV, with link failures at $t = 0.8245s$ and $t = 3.5315s$ from test-suite-sack.tcl in the `ns`-2 distribution.

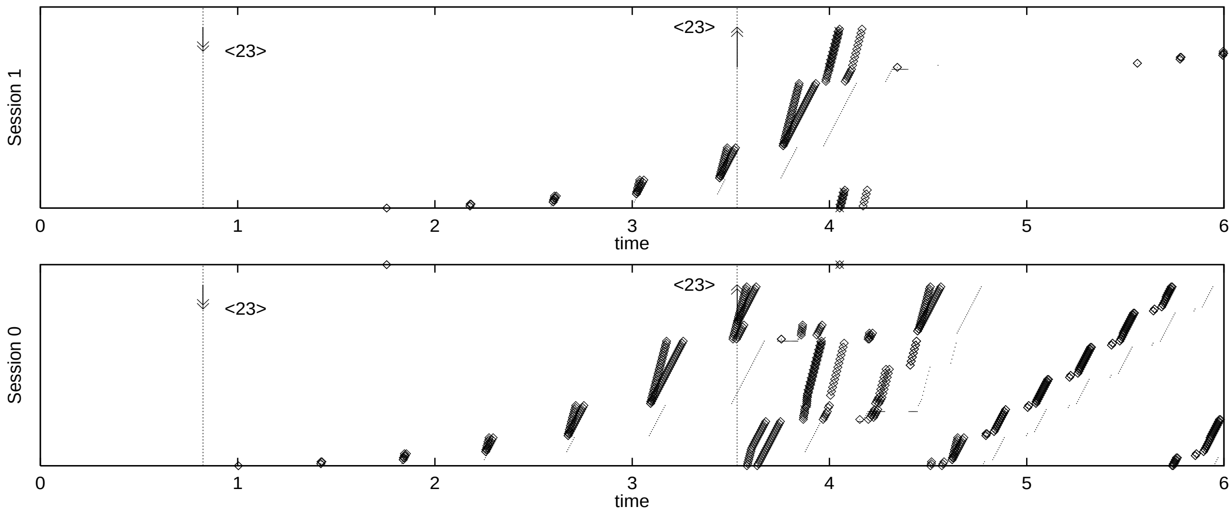


Figure 6: Effect of interleaving packets on TCP SACK

Figure 6 only experiences drops due to congestion at time $t \approx 4.0s$.

This simulation points to a slightly less conservative behaviour in TCP SACK about which packets it should retransmit. The source waits for three dup-acks before retransmitting the first packet, as do TCP Tahoe and TCP Reno. However, based on the selective acks from the receiver, the source infers loss of packets not explicitly acked and retransmits them, leading to possibly unnecessary retransmits of packets that are not actually lost. We saw this earlier in Figure 6, when it retransmit all six packets that were interleaved at $t \approx 3.5s$. In contrast, TCP Tahoe, which also retransmits packets that are not likely lost, is controlled by slow start; TCP Reno goes into fast recovery assuming that only one packet has been dropped.[6]

In summary, we have shown that studying TCP over dynamic topologies can reveal interesting aspects of TCP. An important consequence of dynamic topologies that we have highlighted is the extensive reordering of packets that can arise due to a small topology change. It is important to recognize that the results in this section are not necessarily a function of the size of the topology. Packet interleaving occurs in the immediate vicinity of a topology change, and this localized re-ordering is maintained all the way to the end-to-end protocol operating at the edges of the network. The ability to study such behaviours is very useful in protocol design.

3 Scalable Reliable Multicast

Unicast protocols, such as TCP, are only affected by changes in the topology if the changes occur on the path between the two end-points. In contrast, multicast protocols interact with multiple parties simultaneously and so involve a higher number of links. Therefore, the likelihood is greater that some of the paths in the source's multicast tree are unstable at any given time. In addition, the instability in any portion of the multicast tree may affect many members of the group because of the collaborative adaptive algorithms used. Therefore, it is even more critical to study aspects of end-to-end multicast protocols, such as error recovery or congestion control mechanisms.

[6] It is generally the case that, if multiple packets in a window have been dropped, the TCP Reno sender ultimately waits for a retransmit timeout, and the initiates slow start [3].

In this section, we evaluate a reliable multicast protocol: Scalable Reliable Multicast (SRM). In SRM, "active sources" in a multicast group periodically send data to the group, as specified by the application. Each unit of data is identified by the tuple $\langle$source id, message sequence number$\rangle$; this corresponds to the notion of "naming in Application Data Units model" used in [4]. The source tags each unit of data with its unique identifier. In addition, every member of the group periodically sends session messages specifying the amount of data it has received from each source. Loss can be detected either through receipt of data from the source, or through receipt of a session from some member of the group. A group member that detects a loss can send out a request for that lost data. The error recovery mechanism in SRM is receiver-oriented; therefore, any group member that has the requested data can respond to that retransmission request. It is not necessary that the original source of a particular data unit be the one to respond to requests for retransmission of that data. In the rest of this paper, we use packets as a synonym for data units.

In order to avoid multiple simultaneous requests by all nodes that detect the loss of a particular packet, each node, n_i that detects the loss will set a random timer in the interval $[C_1 d_s, (C_1 + C_2) d_s]$, where C_1 and C_2 are two protocol request parameters, and d_s is n_i's distance to the source of that packet. n_i will send out its request when the timer expires. If n_i hears a request for that packet before its timer expires, it will cancel its timer. In either case, it will reschedule its timer using exponential backoff. Since duplicate requests from other nodes can mislead a node into backing off an already-backed-off timer, each node also delineates an *ignore-backoff* interval, during which it will ignore requests from other nodes. When it finally receives a repair, the node will set a hold-down timer, with a value of $3d_s$, so that it does not attempt to aggressively schedule and send a repair in response to a duplicate (or ill-timed) request.

Likewise, each node n_j that can send a repair in response to a request will set a random timer in the interval $[D_1 d_r, (D_1 + D_2) d_r]$. D_1 and D_2 are two protocol repair parameters, and d_r is n_j's estimate of its distance to the n_i that sent the request. n_j sends out a repair message when its timer expires. If it receives a repair before its timer expires, then its cancels the timer. In either case, n_j sets a hold-down timer, with a value of $3d_r$, during which time it will ignore all requests for that packet, so that it does not attempt to send out a second repair in response to a duplicate (or ill-timed) request.

We consider two mechanisms by which values for the timer parameters, C_1, D_2, D_1, and D_2, are chosen. In the fixed timer mechanism, all of the nodes use identical preset values. Fixed timers are useful to illustrate the behaviour of the protocol. In practise, we need a mechanism to adapt to the observed delay and loss characteristics exhibited by the network. With adaptive timers, each node, n_i, that detects a loss adjusts its request parameters based on the number of duplicate requests that it receives, as well as the average delay between detecting the loss and hearing the request expressed in units of its *rtt* to the source, d_s. Similarly, each node, n_j, that can repair a loss adjusts is repair parameters based on the number of duplicate repairs that it receives, as well as the average delay between receiving a request and sending a repair, normalized in units of its *rtt* to the requestor, d_r. The adaptive timer mechanism will tolerate a duplicate request (repair), as long as the delay is within one *rtt* to the source (requester).

The terms, C_1 and D_1, are called the deterministic components of the timer mechanisms in SRM. Each node has to set its timer value to at least $C_1 d_s$ (or $D_1 d_r$) from the time it detects a loss (receives a request). If C_2 and D_2 are set to 0, then the C_1 and D_1 components ensure that nodes closer to the loss send out requests and repairs. Hence, this component helps distinguish between, and favour those, nodes that are nearer to the loss. By contrast, when there are multiple nodes that are equidistant to the loss, then the nodes should use probabilistic choice in order to decide which of the nodes will send the first request or repair. The terms, C_2 and D_2, specify the maximum range of the interval in which the timers can be set, and hence are called the probabilistic components of the timer mechanisms in SRM.

Earlier, we had said that each node periodically sends session messages that contain information about the latest message from each source that it has received. In addition, nodes use these session messages to estimate their distances to each other. Each node, m_i, time-stamps every session message it sends. In addition, for every other group member, m_j, that m_i is aware of, m_i advertises the sequence number of the last session message it received from m_j, as well as the time that m_j sent that session message, and the time that m_i received it. m_j can use this information to estimate its distance to m_i.

In the remainder of this section, we will present our simulation results as part of our protocol evaluation. We evaluate the protocol in three different ways: on a stable topology, on a dynamic topology that is always connected, or on a dynamic topology that is occasionally partitioned.[7] The first, evaluation using a stable topology, is a base case evaluation (Section 3.1). We use this to illustrate the protocol behaviour, as well as to define our metrics. We then study the same simulation configuration, when the topology is made dynamic (Section 3.2). This evaluation measure helps us understand the characteristic of the deterministic component of the timer functions. Finally, we look at the performance of the protocol under partition and partition healing, as well as the impact of the protocol on the network (Section 3.3).

3.1 Base Case: Analysis with Stable Topologies

In this section, we will evaluate SRM in a stable topology; *i.e.*, the topology is not dynamic. This serves to briefly review the protocol behaviour, as well as to illustrate the different data presentation methods that we will use in the following subsections.

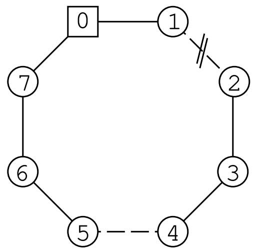
We use the ring topology shown in Figure 7. The bandwidth of each of the links in the topology is $1.5Mbps$. The delay for all but one of the links is $10ms$; the delay for Link $\langle 4, 5\rangle$ is $80ms$. Link $\langle 4, 5\rangle$ is called the fallback link, and will only be used if one of the other links in the topology has failed.

This topology is a very simple model of a network with alternate paths for fallback (or fallback paths). Such fallback paths are used in the event of failure of some of the other paths in the network. When the topology is stable, our topology resembles a string topology [4]. These topologies stress the deterministic component of the SRM timers. There are other topologies that stress other aspects of the protocol, or model more realistic topologies. We have left the studies on these other topologies for future study.

As before, the simulations in this section use our simple Distributed Bellman-Ford routing protocol for unicast routing. The multicast routing protocol is a dense mode variant of DVMRP [22]. For all sources in every multicast group, each node n_i computes a parent-child relationship graph; the graph specifies whether n_i is upstream or downstream of each of its neighbors in the shortest path spanning tree of any given source in a multicast group. n_i uses the receipt of a unicast route update from a neighbour as a signal to recompute its parent-child relationships. The multicast algorithm is a flood and prune algorithm. Therefore, n_i periodically floods multicast packets to all of its neighbours that are downstream to the packet source. Neighbours that do not have any members in the group will send a prune back to their upstream neighbour. Prune state associated with each of its neighbours times out every $0.5s$.

In order to keep our current analysis simple, we have assumed a group density of one, *i.e.*, there is an SRM agent at every node in the topology. Different aspects of the protocol are dependent on group density; in this paper,

[7] The simulation code for the experiments in this section are available at http://www.isi.edu/ ~kannan/ code/ impact.tar.gz. The scripts used for processing this data are separately available at http://www.isi.edu/ ~kannan/ code/ srm-scripts.tar.gz.



In the topology,
— the source is located at Node 0,
— all links are $1.5Mbps$*,*
— all links except Link $\langle$*4, 5*$\rangle$ *have* $10ms$ *delay,*
— the delay on Link $\langle$*4, 5*$\rangle$ *is* $80ms$*,*
— Link $\langle$*1, 2*$\rangle$ *is dynamic,*
— Links $\langle$*1, 2*$\rangle$ *and* $\langle$*5, 4*$\rangle$ *periodically drop data packets from Node 0.*

Figure 7: Cyclic topologies used in the study of reliable multicast

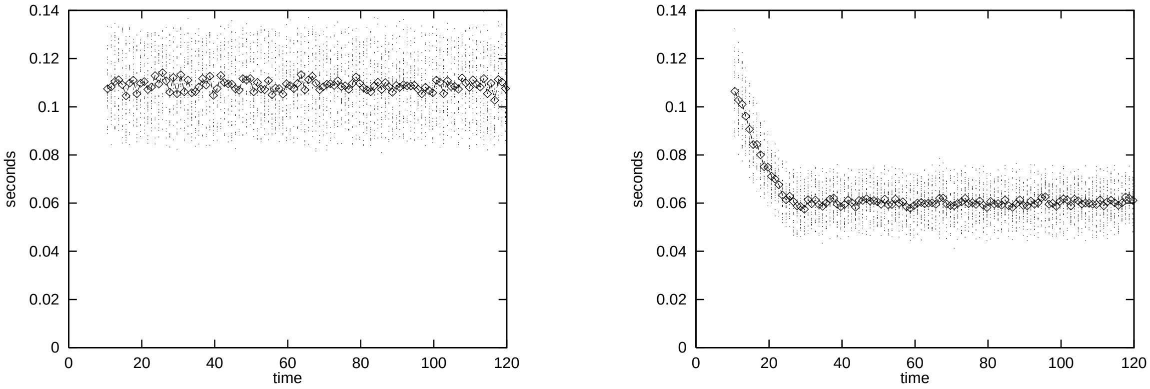
we explore the behaviour of the timer mechanisms in the protocol, which are a greater function of the topology, than the group density. Hence, our results are not invalidated by this simplifying assumption. As a notational convenience, we do not distinguish between a node in the topology and an SRM agent running on that node in the topology.

A constant bit rate source generates two packets per second, and is attached to Node 0. We generate periodic loss of the data stream to observe the behaviour of the adaptive timers. Links $\langle 1{,}2\rangle$ and $\langle 5{,}4\rangle$ drop every other packet from the source. This approach allows us to precisely quantify the loss. In this configuration, only Nodes 2, 3, and 4 will experience all of the data losses and therefore attempt error recovery; the other nodes in the topology will attempt to repair the loss. The losses on the two links are independent of each other.[8] The data rate is set so that each loss and recovery will be complete before the source sends the next packet.

Each of our simulations runs for $200s$. In order to let the initial routing become quiescent, as well as to let the distance estimation algorithms in SRM converge, losses begin at $t = 10s$; we start our evaluations after $t = 20s$. For clarity, we only show the results till $t = 120s$ in our plots. We ran each experiment 31 times, with a different seed for the random number generator in our simulator each time. Unless otherwise specified, our results are a plot of the average of the 31 runs. These plots also include the results from each of the individual runs as tiny dots to illustrate the distribution.

"Recovery delay" is the time difference between when a node detects a lost data unit to the time it actually receives the repair from another node. Figure 8 shows the average recovery delay for the nodes in the topology to recover from each loss. The x-axis corresponds to the approximate time that the loss was detected by any of the nodes. This allows us to assign a unique time stamp to that event; we use this idea in later evaluations to correlate other events in the simulation, such as topology change events. The y-axis indicates the recovery delay. In Figure 8(a) all of the nodes use fixed timers; In Figure 8(b) all of the nodes use adaptive timers. From the figures, we see that fixed timers has a constant recovery delay; adaptive timers gradually improve the recovery delay at all the nodes experiencing the loss.

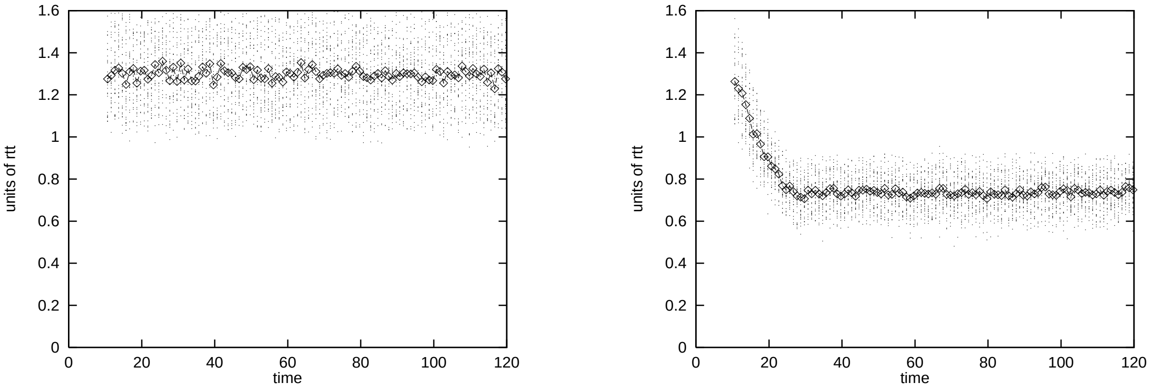
[8] In particular, note that we are using dense-mode multicast, based on periodic flood-and-prune. In our topology, the fallback link, Link $\langle 4{,}5\rangle$, is not usually used. Node 5 will periodically flood data to its neighbour, Node 4 across this fallback link. Node 4 sends a prune whenever Node 5 sends data packets across the fallback link; (the exception of course, is when the link must be used because some other link in the topology has failed). Link $\langle 4, 5\rangle$ will drop alternate packets seen in this periodic flood as well, and hence, over a period of time, there may not be a one-to-one correlation between packets dropped on Link $\langle 1{,}2\rangle$ and Link $\langle 4{,}5\rangle$.



(a) Fixed Timers

(b) Adaptive Timers

Figure 8: Average recovery delay per loss



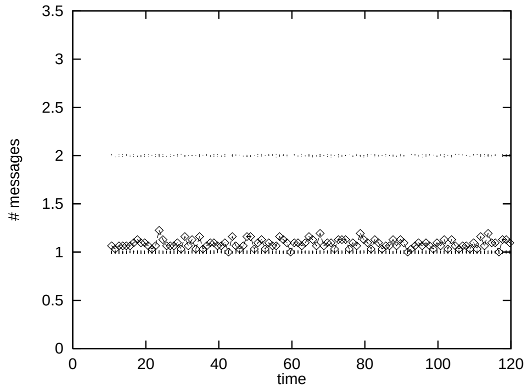
(a) Fixed Timers

(b) Adaptive Timers

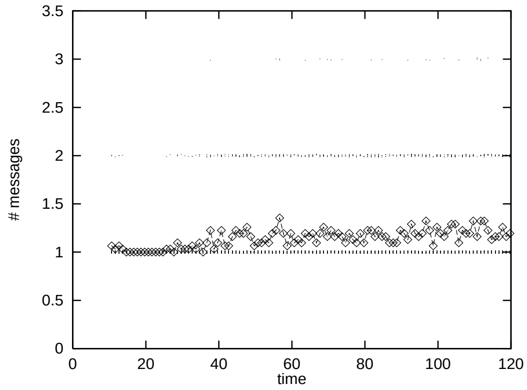
Figure 9: Normalized recovery delay, expressed in units of *rtt* per loss

Since the recovery delay (and by extension, the average recovery delay) is a function of which node initiates the request, and which other node performs the repair, this metric can have high variability. An alternate measure, used in [4], is to normalize the recovery delay at a node, and express it in units of that node's *rtt*. In particular, for any given loss, we determine the last node to receive the repair, and normalize the delay experienced by that node its estimate of the *rtt* to the source. This is a measure of the worst case recovery delay experienced by these nodes. Figure 9 shows the plots of this metric for Fixed and Adaptive timer mechanisms. From Figure 9(a), we see that, when using Fixed timer mechanisms, the worst case recovery delay for any node is over 1.2*rtt* to the source for that node. On the other hand, we can see from Figure 9(b) that the delay is within 1*rtt* for that node, when the nodes use adaptive timers.

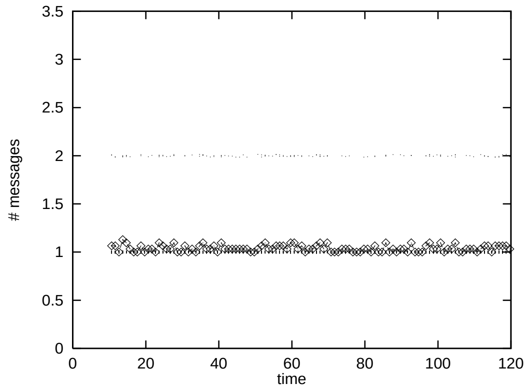
These figures illustrate some of the original motivation in [4] for developing adaptive timer mechanisms, *i.e.*, to reduce the recovery delay, at the expense of a marginal increase in the number of duplicate requests. We therefore look at the number of requests and repairs sent by each of timer mechanisms.



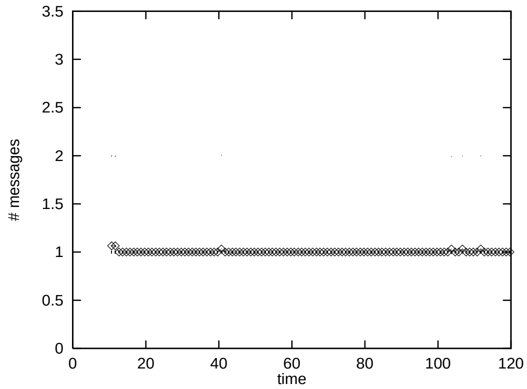
(a) # of requests sent in fixed timer mechanisms



(b) # of requests sent in adaptive timer mechanisms



(c) # of repairs sent in fixed timer mechanisms



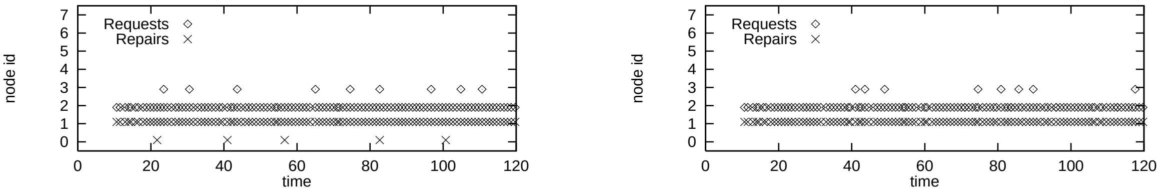
(d) # of repairs sent in adaptive timer mechanisms

Figure 10: Request and repair counts per loss

We can characterize a protocol's performance as a count of the number of request and repair messages that are sent for each loss. Figure 10 shows these plots of request and repair messages for fixed and adaptive timers. In order to show all of the different points in the distribution, we plot the distribution as jittered dots. From Figure 10(a) and 10(c), we can see that fixed time mechanism send few duplicate requests or repairs. On the other hand, adaptive timers send slightly higher number of requests (Figure 10(b)) in order to reduce the recovery delays following a loss. Figure 10(d) tells us that adaptive timer mechanisms are more consistent in not sending duplicate repairs than fixed timers in our topology.

From the figures for normalized recovery delay (Figure 9(b)) and request counts (Figure 10(b)), we see that adaptive timer mechanisms tends to admit some number of duplicate requests and repairs in order to ensure that the recovery delay is within expected bounds. For the rest of this paper, we will evaluate the behaviour of the adaptive timer mechanisms in response to topology change events.

As an aside, Figure 11 shows the traces from two specific simulations; each is a plot of the nodes that send the individual reques and repair messages for each loss. Each simulation is run using the same seed, Figure 11(a) shows the results using fixed timers, and Figure 11(b), using adaptive timers. In each case, we see that Node 2 sends out most of the requests and Node 1 does most of the repairs for fixed timers, which is as we expect in a string

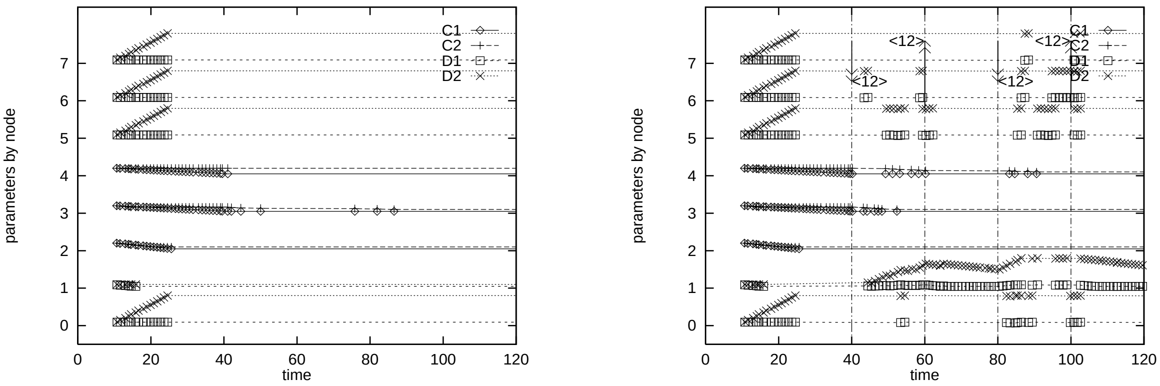


(a) Fixed timers

(b) Adaptive Timers

Note that this plot is the result from a single simulation run, and does not indicate any distributions.

Figure 11: Messages sent by each node for each loss



(a) Stable Topologies

(b) Dynamic Topologies

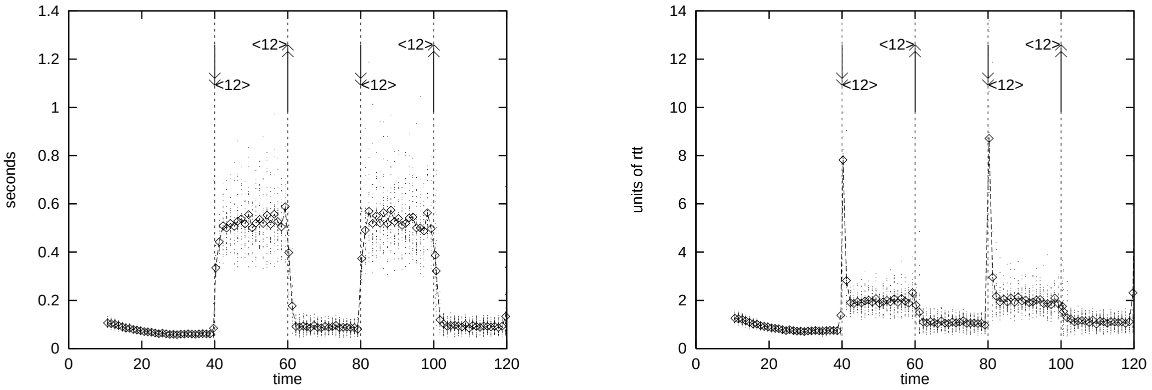
Note that this plot is the result from a single simulation run, and does not indicate any distributions.

Figure 12: Parameter adaptation by each node: Adaptive timers

---

topology. With adaptive timers, these nodes send out all of the requests and repairs, while Node 3 also sends out the occasional duplicate requests. A final measure of protocol behaviour that could help improve our comprehensibility of the protocol is a plot of the request and repair parameters at each of the nodes. Figure 12(a) shows this plot for one run of the simulation. The y-axis shows the parameters for each node offset by an appropriate amount. Note that each node is only involved in request or repair for each loss. From the figure, we see that Node 1 reduces its D_1 and D_2, so that it is always the one to send out repairs. All other nodes that can repair a loss increase their D_2, so that they would rarely send out a repair message. Node 2 reduces its C_1 and C_2, so that it always sends out a request message. By reducing its parameters significantly, and rather quickly, Node 2 ensures that the recovery delay is already fairly small. Over time, Node 3 adapts its parameters to enable it to send an occasional duplicate request. This explains the occasional duplicate request from Node 3 that we see in Figure 11(b).

We also notice from the figure (Figure 12(a)) that in stable topologies, the nodes adapt to the topology very quickly, and subsequently exhibit very little adaptivity. In contrast, Figure 12(b) shows the same parameter adaptation for the same simulation, when the topology is dynamic. We notice immediately that some of the nodes are continually



(a) Average recovery delay

(b) Normalized recovery delay

Figure 13: Recovery delay per loss: Adaptive timers over dynamic topologies

adapting to every change in the topology. In the following section, we will evaluate protocol behaviour over dynamic topologies.

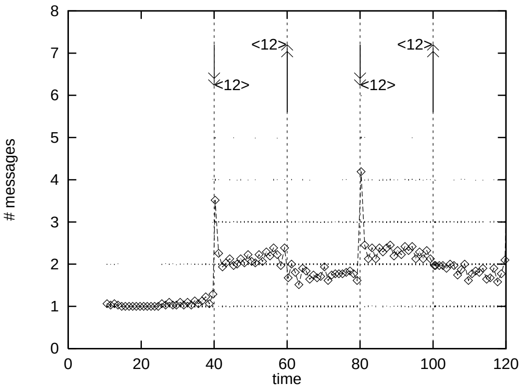
3.2 Evaluation of the Adaptive Timer Mechanisms

In this section, we repeat the experiments of the earlier section, when running SRM over dynamic topologies. For the sake of brevity, we concentrate on adaptive timer mechanisms in the remainder of this paper.

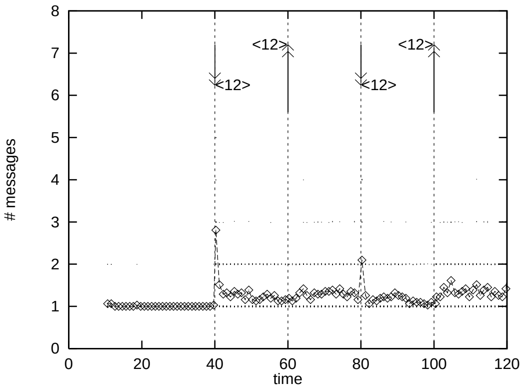
The choice of the network dynamics model for these experiments is guided by the following criteria: (a) the topology changes must occur in such a manner that, regardless of the topology, the same set of nodes experiences the losses and the same set of nodes can repair the loss; and (b) the interval between two successive topology changes should be sufficient to let the nodes adapt to the new topology. This permits us to study the adaptation of the timer mechanisms. In addition to the above two criteria, it is also useful to have topology changes occur at predictable intervals. While this last criteria does not model any operational characteristic, it helps us isolate the effect of each topology change from the next. Therefore, for the rest of the experiment in this paper, we use a deterministic network dynamics model, one in which Link $\langle 1, 2\rangle$ periodically fails and recovers every $20s$. The first event is the link failure at $t = 40s$.

Link $\langle 1, 2\rangle$ will fail at $t = 40s$ and $t = 80s$ in our plots, and recover at $t = 60s$ and $t = 100s$. During the intervals, $[40s, 60s]$ and $[80s, 100s]$, the alternate higher delay path through Link $\langle 4, 5\rangle$ is used. In our plots, we show the time at which topology changes occur as a vertical line at the appropriate instant in time. Markers on the line (seen towards the top of the plot) indicate whether the link fails (goes down) or recovers (comes up); a label indicates the link that has changed state.

Figure 13 shows the plots of average and normalized recovery delays for the nodes experiencing loss in the topology. In particular, Figure 13(a) illustrates the expected increase in the average time to recover from a loss every time the alternate path through Link $\langle 4, 5\rangle$ is used. In these intervals, we see the average delay increase from $\approx 0.1s$ to $\approx 0.8s$. Correspondingly, Figure 13(b) indicates that the normalized delay increases from ≈ 1*rtt* to ≈ 2*rtt*.



(a) # of requests sent in adaptive timer mechanisms



(b) # of repairs sent in adaptive timer mechanisms

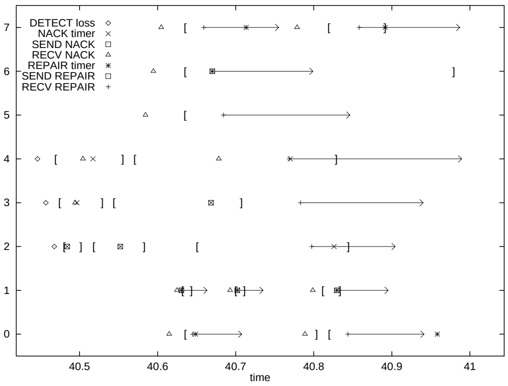
Figure 14: Request and repair counts per loss: Adaptive timers over dynamic topologies

By design, the increase in normalized delay will cause the nodes to attempt to send more duplicate requests in an attempt to reduce their recovery delay. Figure 14(a) confirms that the nodes send more than 2 requests in the intervals when the normalized delay increases over 2*rtt*, and decreases to sending about 1.5 requests on average, at other intervals. In contrast, the number of repairs sent is about 1.5, and never changes significantly (Figure 14(b)).

While the results in terms of the duplicate requests and repairs, and recovery delays are as expected, we can observe two anomalies: (1) exaggerated spikes at $t = 40s$ and $t = 80s$. corresponding to the instants when Link $\langle 1, 2 \rangle$ fails, and (2) increases in the number of repairs sent at the same instants in time. The first anomaly is an artificial consequence of protocol behaviour operating over dynamic topologies. The second is a real consequence of protocol behaviour. We discuss each of these in turn.

The anomalous spikes in the normalized delay that occur when Link $\langle 1, 2 \rangle$ fails are an artifact of operating over dynamic topologies. The reason for these spikes is that, just after the link failure, the time to recover increases for all the nodes. However, the distance estimates at each of the nodes is still the original, much smaller estimate. In particular, in our ring topology, the node that will get the repair last is the node that had the shortest distance estimate of all the three nodes before the failure. All three factors above, contribute to the exaggerated spikes in the normalized recovery delay at the instant of the link failure.

Since the adaptive timer mechanisms attempt to optimize normalized delay by sending duplicate requests, we might expect that the spikes in the number of requests at $t = 40s$ and $t = 80s$ in Figure 14(a) are reasonable. However, this does not explain the corresponding peaks in the number of repairs at those instants (Figure 14(b)). The reason for these becomes apparent when looking at a detailed trace of a single request and repair cycle following a link failure (Figure 15). Recall that, just after Link $\langle 1, 2 \rangle$ fails, the nodes have very short distance estimates to each other; this estimate is based on information learned before the topology change. Therefore, each of the nodes will set its timers too short. Nodes experiencing a loss will set multiple "rounds" of requests, sending a request after each round, and setting their timers again, and again, until they receive the repair. In a similar manner, nodes that can send the repair use correspondingly short distance estimates. They will set short hold-down times following the repair, and send multiple repairs, corresponding to each of the requests sent. We can see this clearly in the trace of Figure 15. The figure shows the time lines at all of the nodes following a failure of Link $\langle 1, 2 \rangle$ at $t = 40s$ Nodes 2, 3, and 4 go through at least two rounds of request, and they send three requests between them. Nodes 0, 1, and 7 set their hold-down timers too short, and set and send multiple repairs. Node 0 in particular

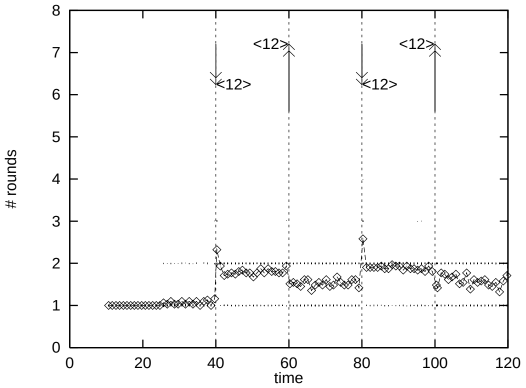


This figure shows a possible sequence of events following a link failure, in which nodes set and send multiple rounds of requests and repairs. The x-axis shows the time; along the y-axis, we plot the timeline for each node, and the sequence of events at that node.

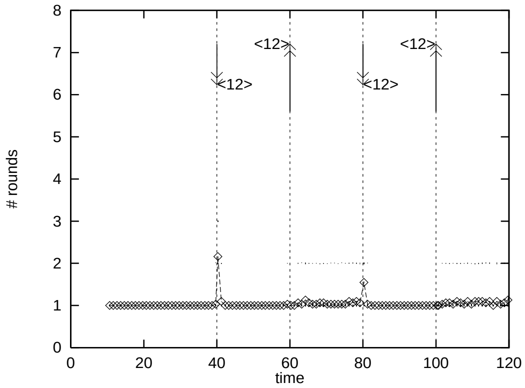
The arrows indicate the duration of the hold-down timer following the sending or receipt of a repair message. The square brackets show the interval from which a node sets its nack or repair timer.

In this topology, Nodes 2, 3, and 4 send requests, and the other nodes send repair messages.

Figure 15: Trace of messages for a single loss following a link failure



(a) # rounds of request



(b) # rounds of repair

Figure 16: Maximum request and repair rounds per loss: Adaptive timers

sends three repairs corresponding to each of requests it receives.

Since the nodes execute multiple request and repair rounds, following a topology change, Figure 16 plots the maximum rounds of request and repair executed by any node. From Figure 16(a), in dynamic topologies the nodes execute multiple rounds of requests; ≈ 1.8 request rounds when Link $\langle 1, 2\rangle$ is down, ≈ 1.5 request rounds at other times. As expected, we also see the transient rise in the number of rounds of repair, that we discussed in the earlier paragraphs occurs at $t = 40s$ and $t = 80s$ (Figure 16(b)). However, we also observe a secondary phenomenon in this figure; the nodes execute more repair rounds when all the links in the topology are up, and the distances between the nodes are optimal, *i.e.*, in the time intervals $[60s, 80s]$ and $[100s, 120s]$. It is not immediately obvious to us as to why this occurs and requires further investigation; we have deferred this investigation for future work.

We have already observed that it is critical for the error recovery mechanisms to obtain good estimates of their distances to all other members. Distance estimation in SRM is achieved through the exchange of session messages by group members. It takes at least three iterations of session messages for two nodes to find their distances to each other. Our session message frequency in these experiments was one message every second; hence it comes as no surprise to find that the spikes in all of our graphs only last for a couple of losses. Appendix A describes the effect of the frequency of session messages on the loss recovery characteristics of the protocol.

In practise, it is impractical for nodes to be sending such frequent session messages, if only because session messages consume expensive bandwidth; they also incur processing overheads at both the nodes that send it, and the nodes that receive and process the session messages. Therefore, we need to determine the optimal frequency of sending session messages, and balance the requirements of bandwidth for distance estimation against that required for other functions, such as error recovery. In this context, the idea of using scaled session messages such that nodes consume a limited amount of bandwidth when sending session messages, attempt to limit the scope of their session messages, and at the same time designate a representative node to send more frequent session messages to the entire multicast group, bears promise [20].

3.3 Recovery after Partitions

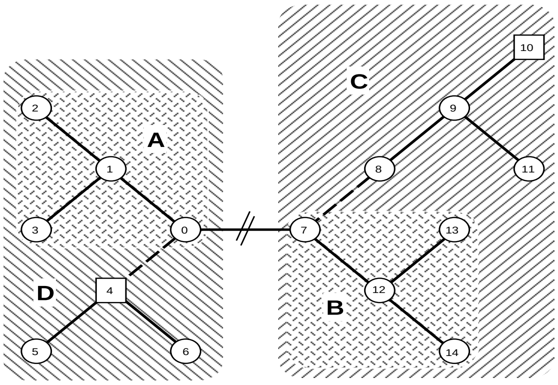
In the previous sections, we considered protocol behaviour when the topology was stable (Section 3.1), and when the topology was dynamic, but continuously connected (Section 3.2). In this section, we study protocol behaviour when the topology is dynamic, and results in occasional network partitions. Partitions and partition healing are more complex situations for multicast protocols. This is because reliable multicast protocols are often designed to continue operation in spite of the partition. Contrast this with unicast protocols where, because of fate-sharing, nodes affected by a partition will make some number of attempts at recovery and then fail. Therefore, it is crucial to study the effect of the partition and healing on reliable multicast protocols.

Before we describe our protocol evaluation of the adaptive timer mechanisms in SRM, we must outline our simulation model that we will use in this section, *i.e.*, the topology, sources, and loss patterns. Our protocol evaluation is based on our classification of the losses that occur due to network partitions. We also address the effect of the protocol behaviour on the network itself.

For this experiment, we need a topology that is susceptible to partitions. We use the tree topology of 14 nodes, shown in Figure 17. In this topology, Link $\langle 0, 7\rangle$ is dynamic. The periodic failure and recovery of Link $\langle 0, 7\rangle$ generates the partition events that are of interest to us. We use the same model of network dynamics as in the earlier section (Section 3.2). Link $\langle 0, 7\rangle$ fails at times $t = 40s$ and $t = 80s$ resulting in partitions. The partition heals when Link $\langle 0, 7\rangle$ recovers at times $t = 60s$ and $t = 100s$. Therefore, the topology is partitioned into two connected components in the intervals $[40s, 60s]$ and $[80s, 100s]$.

The aim of the experiment is to study the impact of network partition and healing on the adaptive timer mechanisms in SRM. Since the timer mechanisms adapt to observed losses, we want a topology in which all the nodes continually receive some data, and there is intermittent loss of some fraction of that data, regardless of whether the network is fully connected or partitioned. This requires a source on either side of the partition, and some number of lossy links that periodically, but continuously, drop data from each of the sources. To satisfy these constraints, we place two constant bit rate sources, each generating two packets per second, one on Node 4, the other on Node 10. Link <4, 0> is configured to drop every other data packets originating from Node 4; likewise, Link $\langle 8, 7\rangle$ is configured to drop every other data packets originating from Node 10.

We can group the receivers in the topology by the pattern of losses they observe for data from each of the sources.



In the topology,
— there are two sources, one at Node 4, the other at Node 10,
— All links have bandwidth $1.5Mbps$, delay $10ms$.
— Link ⟨0, 7⟩ is dynamic,
— Links ⟨4, 0⟩ and ⟨8, 7⟩ periodically drop data packets from Nodes 4 and 10 respectively.

Figure 17: Topology used for evaluation of SRM under partitions

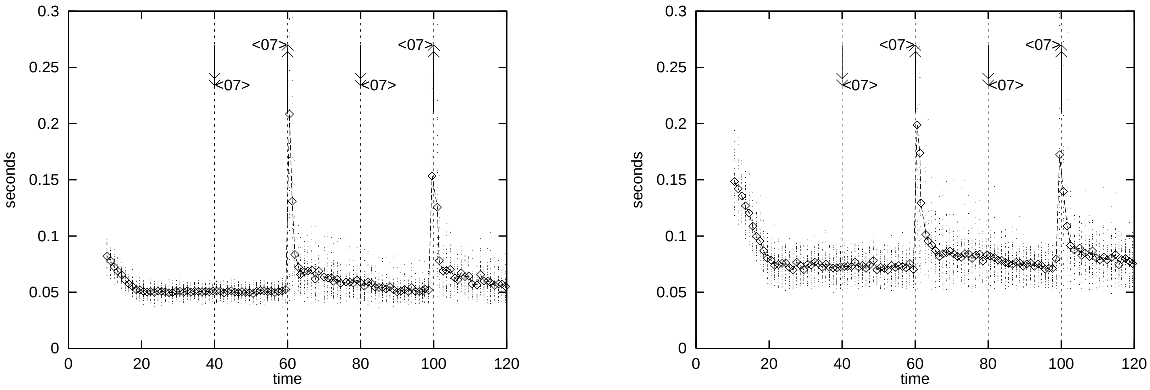
This pattern of loss depends on the location of the receiver relative to the source with respect to the dynamic link. Figure 17 shows the receivers, classified by the type of loss characteristics they experience. Nodes 4, 5, and 6 experience no loss of data from the source at Node 4; likewise, Nodes 8, 9, 10, and 11 experience no loss of data from the source at Node 10.

Nodes 0, 1, 2, and 3 belong to the same component as the source at Node 4, and therefore continually see data and periodic losses from this source at all times, regardless of whether the topology is fully connected or partitioned. We call the loss characteristics observed by this group of nodes, Type A. The average recovery delay for a loss of Type A is the computed over the recovery delays seen by the Nodes 0, 1, 2, and 3 for that loss. Similarly, Nodes 7, 12, 13, and 14 belong to the same component as the source at Node 10; they only see periodic loss of data for data from Node 10. These are Type B losses. The average recovery delay for a loss of Type B is the computed over the recovery delays seen by the Nodes 7, 12, 13, and 14 for that loss.

During a partition, Nodes 7 through 14 will be separated from the source at Node 4. Therefore, they will experience both losses due to partition, and periodic losses due to Link ⟨4, 0⟩ dropping packets. These are Type C losses. The average recovery delay for a loss of Type C is the computed over the recovery delays seen by the Nodes 7 through 14 for that loss. Likewise, Nodes 0 through 7 will be separated from the source at Node 10 during a partition. They will experience loss of data from the source, due to partition as well as periodic loss due to Link ⟨8, 7⟩ dropping packets, when the topology is connected. These are Type D losses. The average recovery delay for a loss of Type D is the computed over the recovery delays seen by the Nodes 0 through 7 for that loss.

We now characterize the performance of the protocol through plots of the average recovery delays for each type of loss. As in the previous sections, our plots show the average of 11 runs of the simulation. The plots also include the results from each of the individual runs (as tiny dots) to illustrate the distribution.

Since Type A and B losses are not affected by network partition, the recovery delay observed by the nodes experiencing these losses will form our base case. Ideally, the partition should have no significant impact on the recovery delay, since the data losses are unrelated to any partition. However, in practise, a flurry of requests and repairs following a partition healing can result in transient congestion (and possible packet losses) within the network,



(a) Type A losses: Source Node 4

(b) Type B losses: Source Node 10

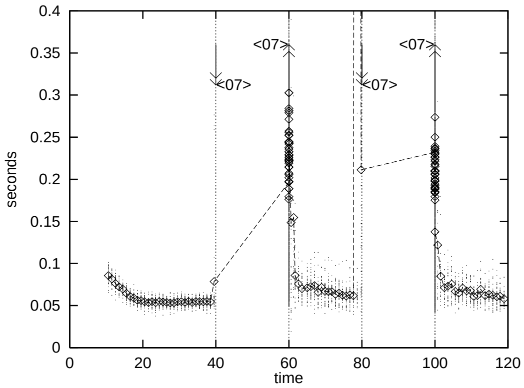
Figure 18: Average recovery delay for losses independent of the partition

which would then increase the average recovery delay for these types of losses. Figures 18(a) and 18(b) shows the recovery delay for Type A and B losses. The figures illustrates the increase in the average recovery delay from $\approx 0.05s$ to $\approx 0.15s$ immediately following a partition healing.

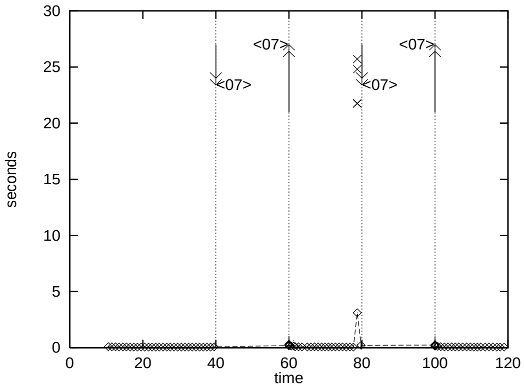
Losses due to network partition are classified as Type C and D losses. However, these types also include periodic loss of data from the source when the network is not partitioned. Figures 19(a) and 19(b) plot the recovery delay seen by nodes experiencing Type C losses for data from the source at Node 4. Both plots are present the same results are different scales. Figure 19(a) shows the detail of the recovery delays occurring in less that $0.4s$, and in particular, the average of all of the simulation runs. Figure 19(b) shows the complete graph, including the distribution obtained from multiple simulation runs; this reveals some of the outliers in some of the simulations. We can study the protocol behaviour in three separate regions: the initial period; the period when the network is partitioned (*i.e.*in the interval $[40s, 60s]$); and the period following the partition healing. We can then analyse the causes for the outliers in the different simulations. Note that the recovery delays seen by nodes experiencing Type D losses for data from the source at Node 10 are identical, and are shown in the plots Figures 19(c) and 19(d). We do not discuss these losses separately in this paper.

In the initial period, we see that the protocol functions normally, until the first link failure event at $t = 40s$. At this point, the network is partitioned and remains partitioned until the link recovers at $t = 60s$. In this interval, none of the nodes can execute any loss recovery, since they have no indication of any loss. All data packets that are to be sent on that link are dropped. After the partition heals, the nodes schedule the recovery of packets lost due to the partition as soon as they detect the loss. This occurs when either a new packet from a source, or a session message from a node in the other component during the partition, is received. Each node schedules the recovery for all lost packets at the same time. However, the observed recovery delay for each of these messages has high variability ($\approx 0.1s$ to $\approx 0.4s$). The same sequence of events repeats during subsequent partition and healing events.

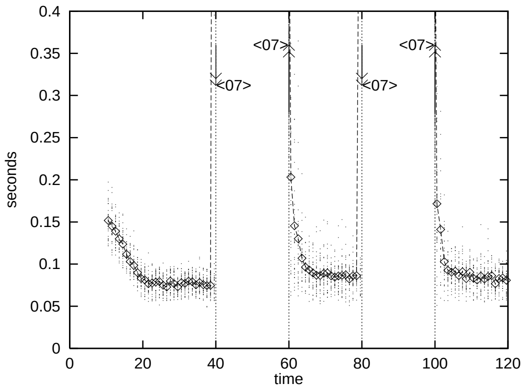
Each of the comple plots (Figures 19(b), 19(d)) reveals a few outliers that raise some interesting design issues. These outliers are shown as 'x' in the two plots and occur immediately before the partitions at $t \approx 40s$ and $t \approx 80s$. In these losses, the nodes detect the loss of a packet corresponding to the time just prior to the network partition. However, the error recovery is stalled due to the partition, and only completes after the partition heals. Hence the recovery time for these losses is of the order of magnitude of the partition. This is an extreme case of recovery delay that can occur in an operational network. While the error recovery is stalled, the nodes attempting



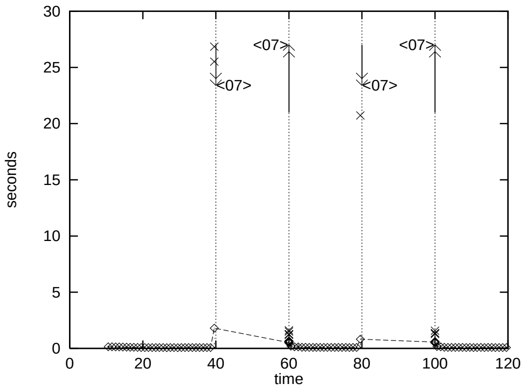
(a) Type C: Source Node 4: Detail



(b) Type C: Source Node 4: Complete Graph



(c) Type D: Source Node 10: Complete Graph



(d) Type D: Source Node 10: Complete Graph

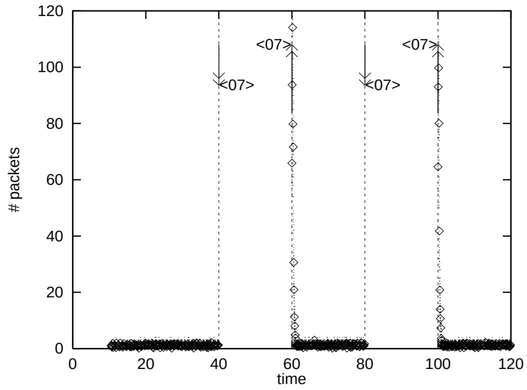
Figures (a) and (b) show the same plots at different scales. Likewise, Figures (c) and (d) show the same plots at different scales.

Figure 19: Average recovery delays seen in the presence of partitions

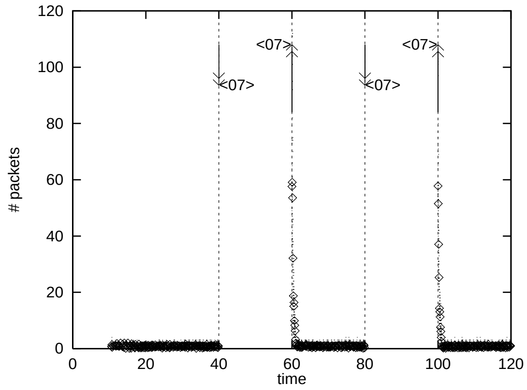
the recovery periodically send out requests, and continually back-off their timers. Therefore, following the partition healing, the nodes are using a significantly larger timer value than is essential for quick recovery. This raises an interesting design question: Could such a node infer the partition healing event based on the arrival of new data or session messages, and use this information to reset its timers to more reasonable values and improve its adaptivity?

We end this section by attempting to quantify the effect of the protocol on the network following a partition or healing event. Figure 20 shows the requests and repairs that are sent on each direction of Link $\langle 0, 7\rangle$. For each run of the simulation, we obtain the number of request and repair packets sent in each direction, and plot the average across the 11 runs. We can see the increase in the number of requests and repairs following the partition healing at $t = 60s$ and $t = 100s$. We also see a corresponding increase in the number of drops on link in either direction. These figures indicate the increase in congestion in the network following a partition healing.

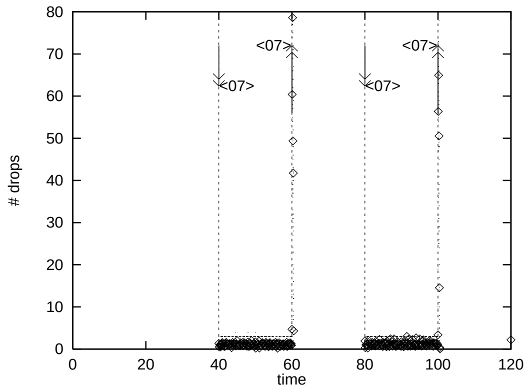
An artifact of the implementation in our simulator is that the congestion control mechanisms are still incomplete. Our plan is to add a simple rate limiter, such that there will be a limit on the aggregate bandwidth used by control traffic. However, this method of congestion control has a number of open issues in the face of partition healing. For



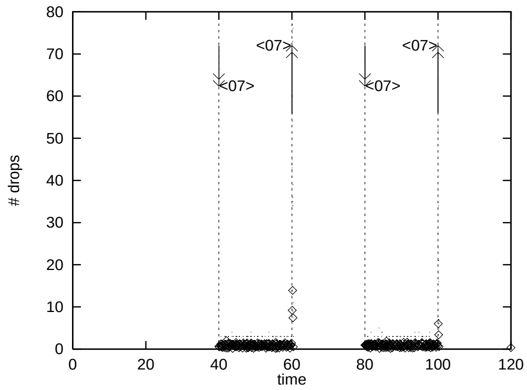
(a) Traffic on Link ⟨0,7⟩



(b) Traffic on Link ⟨7,0⟩



(c) Drops on Link ⟨0,7⟩



(d) Drops on Link ⟨7,0⟩

Plots indicate the number of requests and repairs sent or dropped in each direction of Link ⟨0, 7⟩.

Figure 20: Traffic and drops on Link ⟨0,7⟩

instance, at the instant of partition healing, each node on either side of the partition detects a burst of losses, and independently schedules and send request and repair messages. We leave for future work the question of whether a simple local rate limiting mechanism is sufficient to avoid transient congestion, or whether the nodes should be using a more sophisticated congestion control algorithm.

In summary, we have some preliminary results in our study of SRM over dynamic topologies. Clearly, there is more work to be done in studying other aspects of the protocol in different topologies. Our work is an ongoing exploration of the full range of SRM behaviour, and in this context, the results in this section are only the tip of an iceberg. Several other reliable multicast protocols such as Transport for Reliable Multicast (TRM) [18] and Deterministic Timeouts for Reliable Multicast (DTRM) [6], use timer mechanisms similar to SRM. The results and methods that we have described in this section could apply to these protocols.

4 Conclusions and Future Work

In this paper, we have looked at the impact of network dynamics on TCP and SRM. TCP is a mature protocol that has been extensively studied over a number of years. The advantage of using TCP for a case study is that it is a well-defined protocol, with clearly understood operational behaviour. Reliable multicast, on the other hand, is in a relatively embryonic stage of development. Most of the protocols and mechanisms are still being designed, and there are a number of open issues that are yet to be resolved. This makes it difficult for us to precisely quantify the behaviour of a particular protocol under operational conditions relating to network dynamics. On the other hand, the different reliable multicast protocols have a number of common mechanisms and characteristics. Our goal is to evaluate the behaviour of these common mechanisms in dynamic topologies, and to understand the tradeoffs associated with them.

We conjecture that there will be significant impact of network dynamics in the design of reliable multicast protocols. In particular, these protocols exploit locality in different ways, so as to improve their scalability without compromising their adaptivity. For example, scalable session messages mechanisms [20] choose representative members that will send global session messages on behalf of members in the vicinity. This involves making locality decisions about the topology, to decide the representative, and the group associated with that representative. Network dynamics results in changing the topology, thereby potentially invalidating the locality decisions that were made prior to the topology change.

Other reliable multicast protocols use a combination of mechanisms to exploit locality. These mechanisms fall into four broad classes. (1) Designated representatives can act as proxy on behalf of members in the vicinity [10, 8, 20, 1, 2, 7]. (2) Topologically contiguous members that share a common characteristic, such as errors due to malfunctioning link, or congestion, can form a local group to recover from the effects of that characteristic, (3) some mechanisms form a hierarchy as an additional means of scaling [8, 10, 23], (4) Subcast, a special form of directed broadcast, can be used to aid in the recovery of losses seen by specific portions of the multicast sub-tree [15, 12]. The complexity of each of these mechanisms, as applied in a protocol, directly impacts the adaptability of the protocol.

For each type of mechanism, we would like to address the questions: what is the adaptability of the mechanism to topology change? what fraction of the group is affected by the topology change? what is the impact of protocol adaptation on the network itself? As an example, let us look at the impact of simple topology changes on a representatives mechanism. Posit a sender-initiated reliable multicast protocol that uses these representatives to avoid ack implosion. Consider a topology change that results in the reorganization of a small portion of this multicast tree. What are the parameters that impact the adaptation of this mechanism? What is the impact of the mechanism in relation to the new distance of the representative from the rest of the members of the sub-group? Can the protocol function with the same groups and designated representatives, or should the sub-groups be re-formed? How long does it take for new sub-groups to form, and representatives to be selected? What is the impact on the rest of the multicast group during the time that the sub-groups are being formed anew? What parameters are required for the mechanism to adapt to the partitions? What happens in the case that (a) a representative is separated from its sub-group? or (b) the representative is connected to its sub-group, but the entire sub-group is disconnected from the rest of the multicast group members? How will the multiple disconnected components of the partition continue to function during the partition? What is the behaviour of the protocol after the partition heals? What is the impact on the network when the partition heals? Does the partition healing impact just the nodes affected by the partition, or can it impact the entire group? Can the mechanism be improved to cope with partitions and healing?

We also conjecture that network dynamics will have significant impact on the self-configuration mechanisms used in reliable multicast protocols. These mechanisms are designed to automatically adapt to the environment in which they operate. Examples of such self-configuration mechanisms include local error recovery groups [11], scalable session messages [20], or designating a representative [15, 1, 2]. These self-configuring mechanisms have to cope with group membership changes, as well as topology changes. Therefore, these mechanisms, by definition, should reconfigure when there is a significant change, either in group membership or topology changes. Typically, these mechanisms are evaluated for their responsiveness to group membership changes. Group membership changes and network dynnamics affect protocol operation in different ways. For instance, topology changes can never result in an increase in the group membership; this implies that topology changes model membership changes incompletely. Group dynamics, on the other hand, cannot effectively model the result of a network partition, or a network pathology that results in unidirectional multicast trees, for example. We believe that one of the challenges in the design of reliable multicast protocols is in the design of algorithms that are stable to network dynamics, as well as membership dynamics.

In conclusion, the study of protocols over dynamic topologies is crucial to the design process. There is a lot of work that needs to be done in the evaluation of protocols over dynamic topologies. Our preliminary work reported in this paper serves to illustrate the benefits of such analysis, and points out some of the methods that could be used in such evaluation. Given that protocol designers focus their attention on the questions for which they have the appropriate tools, we hope that this work will encourage new and richer studies of protocol behaviour.

Acknowledgments

We would like to acknowledge the members of the VINT project for their assistance and help with the `ns` simulator. In particular, we wish to thank Tom Henderson (UC Berkeley) and Kevin Fall (LBNL) for their bug fixes and assistance with the TCP SACK code in `ns`; Polly Huang (USC/ISI), for providing the dynamic multicast code that we use in our simulations; and Puneet Sharma (USC/ISI) for his assistance in developing the SRM implementation in `ns`. Mark Handley (USC/ISI), John Heidemann (USC/ISI) and Ching-Gung Liu (USC) provided assistance and feedback on earlier drafts of this document.

A The Role of Session Messages in Loss Recovery

We have already observed that it is critical for the error recovery mechanisms to obtain good estimates of their distances to all other members. Distance estimation in SRM is achieved through the exchange of session messages by group members. It takes at least three iterations of session messages for two nodes to find their distances to each other. Our session message frequency in these experiments was one message every second; hence it comes as no surprise to find that the spikes in all of our graphs only last for a couple of losses. We conducted a series of experiments to characterise the role of the frequency of session messages on error recovery. In this section, we describe the results of this evaluation.

We use the same ring topology used in Section 3; we also use the parameters used for that set of experiments; *viz.*, the same unicast and multicast routing protocols, the same source models, loss models, and group distribution. We conducted experiments for a frequency of sending one session message every $2s$, $3s$, $5s$, and $10s$.[9] Our evaluation

[9] Recall that our earlier experiments (Section 3) consider a session message of $1s$.

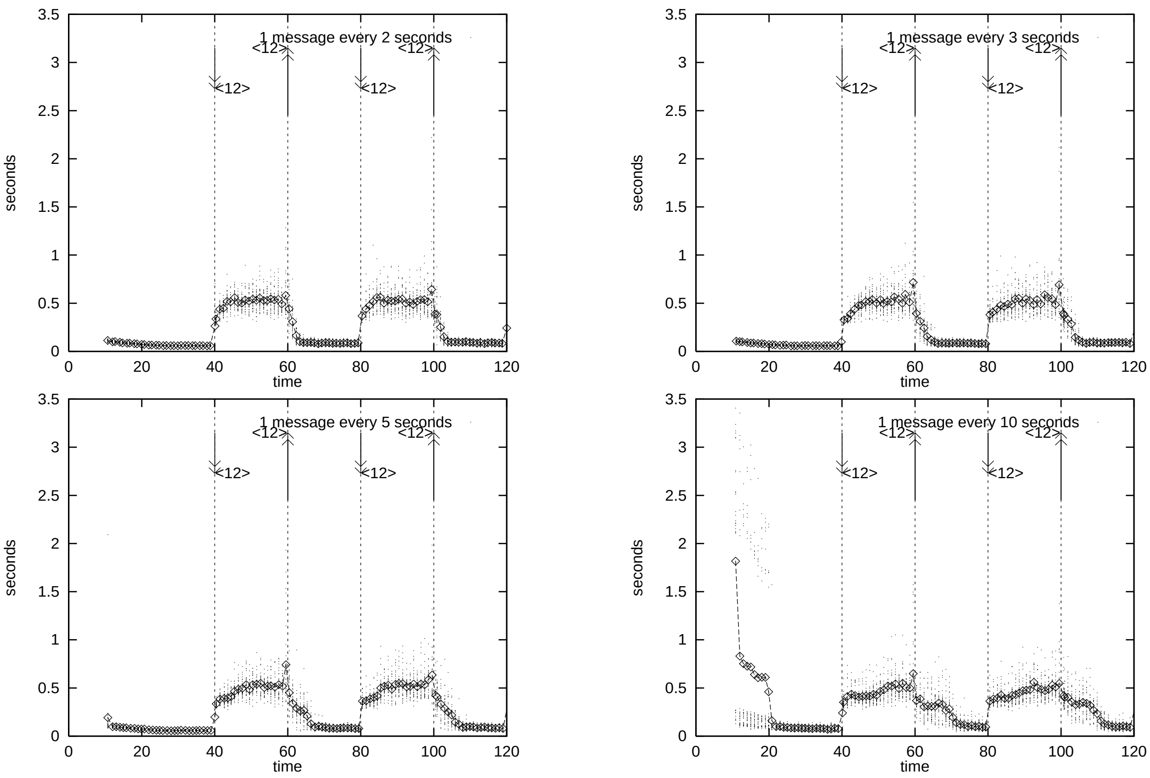


Figure 21: Average recovery delays as a function of the frequency of session messages

of SRM behaviour is based on the recovery delays, the number of duplicate request and repair messages, and the number of rounds of request and repair messages as a function of the frequency of sending session messages.

Figures 21 and 22 show the recovery delays as a function of the frequency of sending session messages. We can make two observations from these plots: first, there is the exaggerated spike in the normalised recovery delays immediately after Link $\langle 1, 2\rangle$ fails (at $t = 40s$ and $t = 80s$), until the nodes estimate the distances in the alternate topology. The duration of the spike is clearly proportional to the frequency of sending session messages.

The second point to note is the decrease in the average recovery time immediately following the failure of Link $\langle 1, 2\rangle$ (at $t = 40s$ and $t = 80s$), and the increase immediately following the recovery of Link $\langle 1, 2\rangle$ (at $t = 60s$ and $t = 100s$). The delay is a "decrease" in the sense that it is not as high as we would expect it to be in the alternate topology (with longer delay paths). This occurs because the nodes are using smaller estimates of distances to each other, and therefore attempt to recover from a loss earlier than they should. The delay following link recovery is an "increase" in the sense that it does not immediately decrease to the expected levels until the nodes have an accurate estimate of the distance. Again, this is because the nodes use the older (and larger) estimate of distances to each other, and hence take longer to attempt to recover from the loss. We can clearly see from the figures that these transients are proportional to the frequency of sending session messages.

As with the recovery delays earlier, we see in Figure 23 that the number of request messages increases to over 2.5–3 immediately following the failure of Link $\langle 1, 2\rangle$ at $t = 40s$ and $t = 80s$, which is what we expect. We see the same type of increase in the number of repair messages (Figure 24), to over two messages, in the same duration. Once again, we can see a clear correlation between the duration of increased duplicate messages, and the frequency of sending session messages.

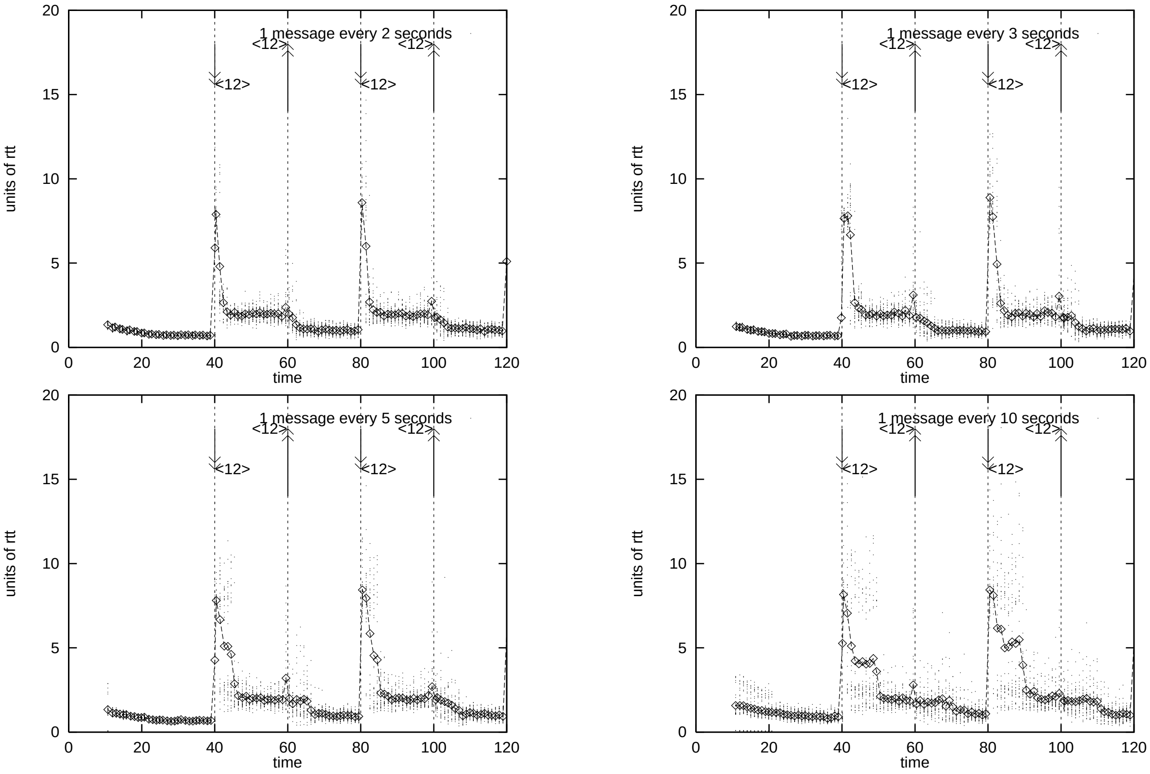


Figure 22: Normalised recovery delays as a function of the frequency of session messages

Finally, we can clearly observe the relationship between the number of rounds of requests and repairs that nodes send out (Figures 25, 26), and the role of distance estimation and the frequency of sending session messages. We can observe this strong correlation in the duration for which nodes execute more than two rounds of request/repair messages following the failure of ink $\langle 1, 2 \rangle$.

References

[1] D. DeLucia and K. Obraczka. A multicast congestion control mechanism using representatives. Technical Report USC-CS TR 97-651, Department of Computer Science, University of Southern California, May 1997.

[2] D. DeLucia and K. Obraczka. Multicast feedback suppression using representatives. In *IEEE Proceedings of the INFOCOM*, 1997.

[3] K. Fall and S. Floyd. Simulation-based comparisons of Tahoe, Reno, and SACK TCP. *ACM Computer Communications Review*, 26(3):5–21, July 1996.

[4] S. Floyd, V. Jacobson, S. McCanne, C-G. Liu, and L. Zhang. A reliable multicast framework for light-weight sessions and application level framing. *IEEE/ACM Transactions on Networking*, 1997. To appear.

[5] R. Govindan and A. Reddy. An analysis of internet inter-domain topology and route stability. In *IEEE Proceedings of the INFOCOM*, April 1997.

[6] M. Grossglauser. Optimal deterministic timeouts for reliable scalable multicast. In *IEEE Proceedings of the INFOCOM*, pages 1425–1432, April 1996.

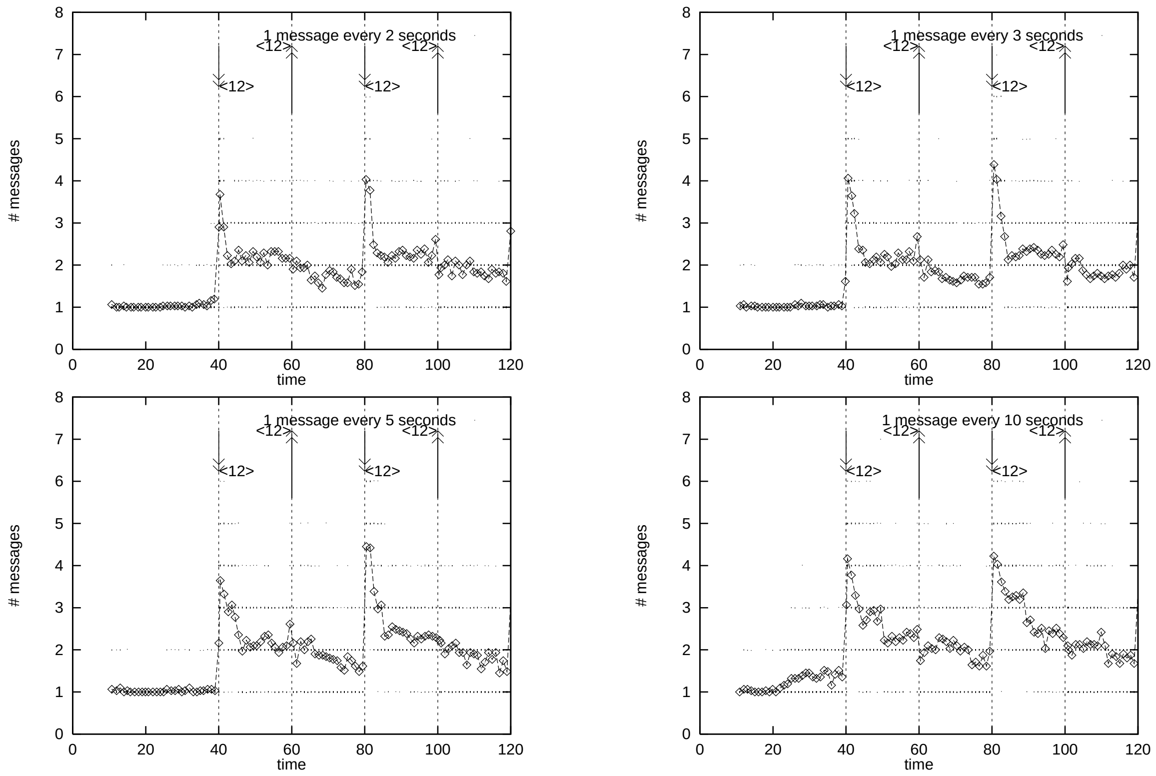


Figure 23: Request message counts as a function of the frequency of session messages

[7] M. Hoffman. Adding scalability to transport level multicast. In *Proceedings of the Third COST 237 Workshop - Multimedia Telecommunications and Applications*, November 1996.

[8] H.W. Holbrook, S.K. Singhal, and D.R. Cheriton. Log-based receiver-reliable multicast for distributed interactive simulation. In *Proceedings of the ACM SIGCOMM*, 1995.

[9] C. Labovitz, G.R. Malan, and F. Jahanian. Internet routing instability. In *Proceedings of the ACM SIGCOMM*. ACM, September 1997.

[10] J.C. Lin and S. Paul. RMTP: A reliable multicast transport protocol. In *IEEE Proceedings of the INFOCOM*, pages 1414–1424, April 1996.

[11] C-G. Liu, D. Estrin, S. Shenker, and L. Zhang. Local error recovery in SRM: Comparison of two approaches. Technical Report USC-CS-TR-97-648, Department of Computer Science, University of Southern California, 1997.

[12] S. McCanne. Router forwarding services for reliable multicast. Note ⟨199704141535.IAA10590@mlk.cs.berkeley.edu⟩ to the Reliable Multicast list ⟨rm@mash.cs.berkeley.edu⟩, April 1997.

[13] S. McCanne and S. Floyd. ns—Network Simulator. http://www-mash.cs.berkeley.edu/ns/.

[14] S. McCanne, V. Jacobson, and M. Vetterli. Receiver-driven layered multicast. In *Proceedings of the ACM SIGCOMM*, pages 117–130, Stanford, CA, U.S.A., August 1996.

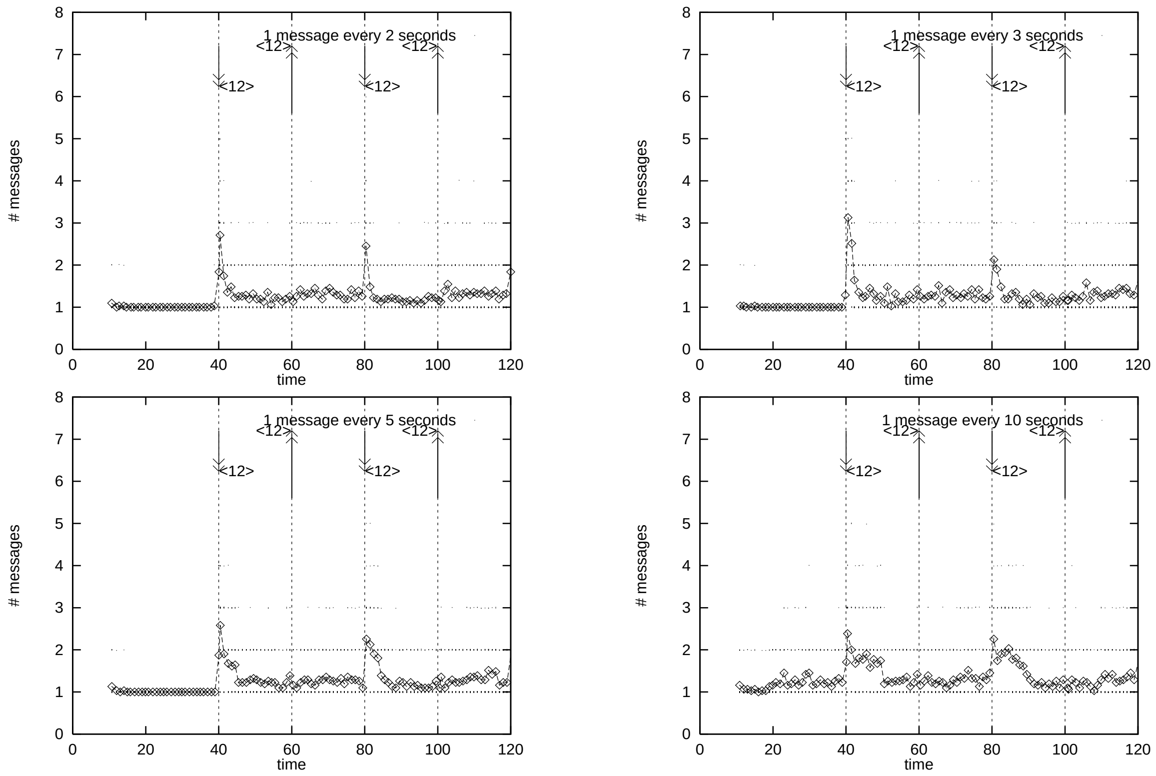


Figure 24: Repair message counts as a function of the frequency of session messages

[15] C. Papadopoulos, G. Parulkar, and G. Varghese. An error control scheme for large-scale multicast applications. http://dworkin.wustl.edu/ christos/ PostscriptDocs/ current.ps.Z.

[16] V. Paxson. End-to-end routing behavior in the internet. In *Proceedings of the ACM SIGCOMM*, August 1996.

[17] J. Postel. *Transmission Control Protocol*, RFC 793 edition, 1981.

[18] B. Sabata, M.J. Brown, and B.A. Denny. Transport protocol for reliable multicast: TRM. In *International Conference on Networks*, Orlando, Florida, U.S.A., January 1996.

[19] A.U. Shankar, C. Alaettinoğlu, K. Dussa-Zieger, and I. Matta. Transient and steady-state performance of routing protocols: Distance-vector versus link-state. *Internetworking: Research and Experience*, 7(1), March 1996.

[20] P. Sharma, D. Estrin, S. Floyd, and V. Jacobson. Scalable timers for soft state protocols. In *IEEE Proceedings of the INFOCOM*, April 1997.

[21] K. Varadhan, S. Estrin, and S. Floyd. Impact of network dynamics on end-to-end protocols: Case studies in reliable multicast. In *IEEE Symposium on Computers and Communication Systems*, August 1998. http://www.isi.edu/ ˜kannan/ papers/ iscc98.ps.gz.

[22] D. Waitzman, C. Partridge, and S.E. Deering. *Distance Vector Multicast Routing Protocol*, RFC 1075 edition, 1988.

[23] R. Yavatkar, J. Griffioen, and M. Sudan. A reliable dissemination protocol for interactive collaborative applications. In *Proceedings of the ACM Multimedia conference*, pages 333–344, 1995.

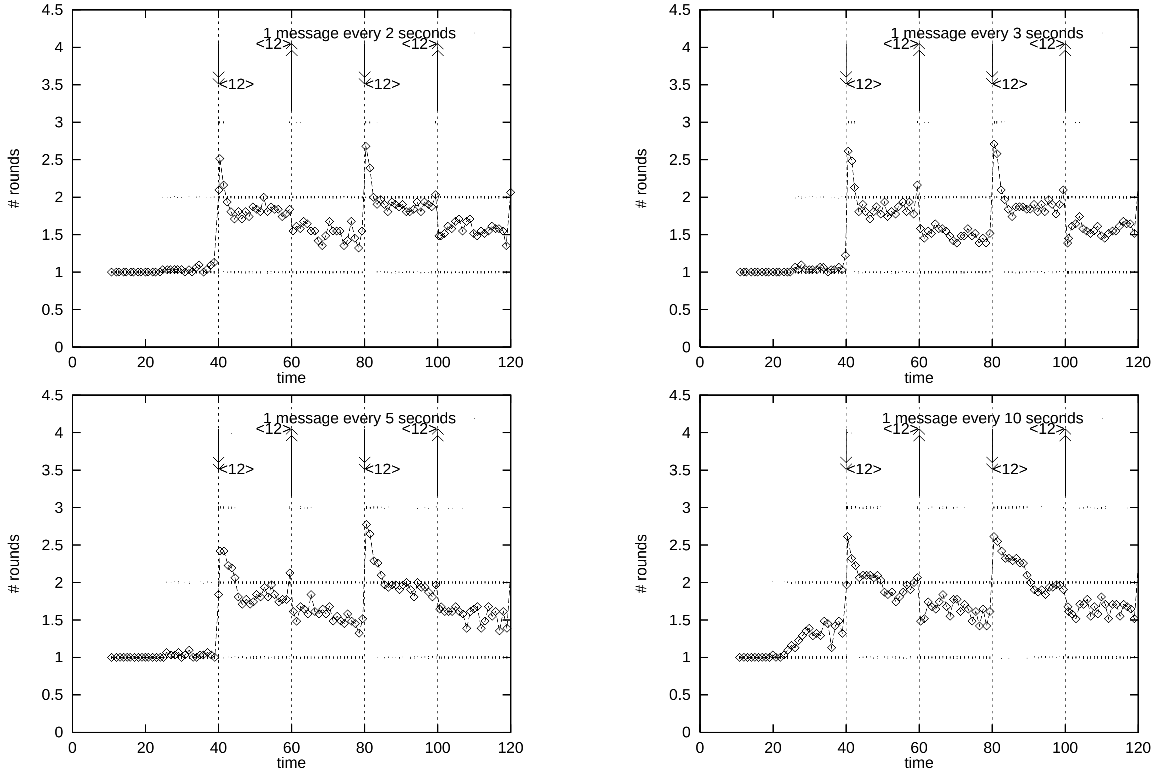


Figure 25: Request message rounds as a function of the frequency of session messages

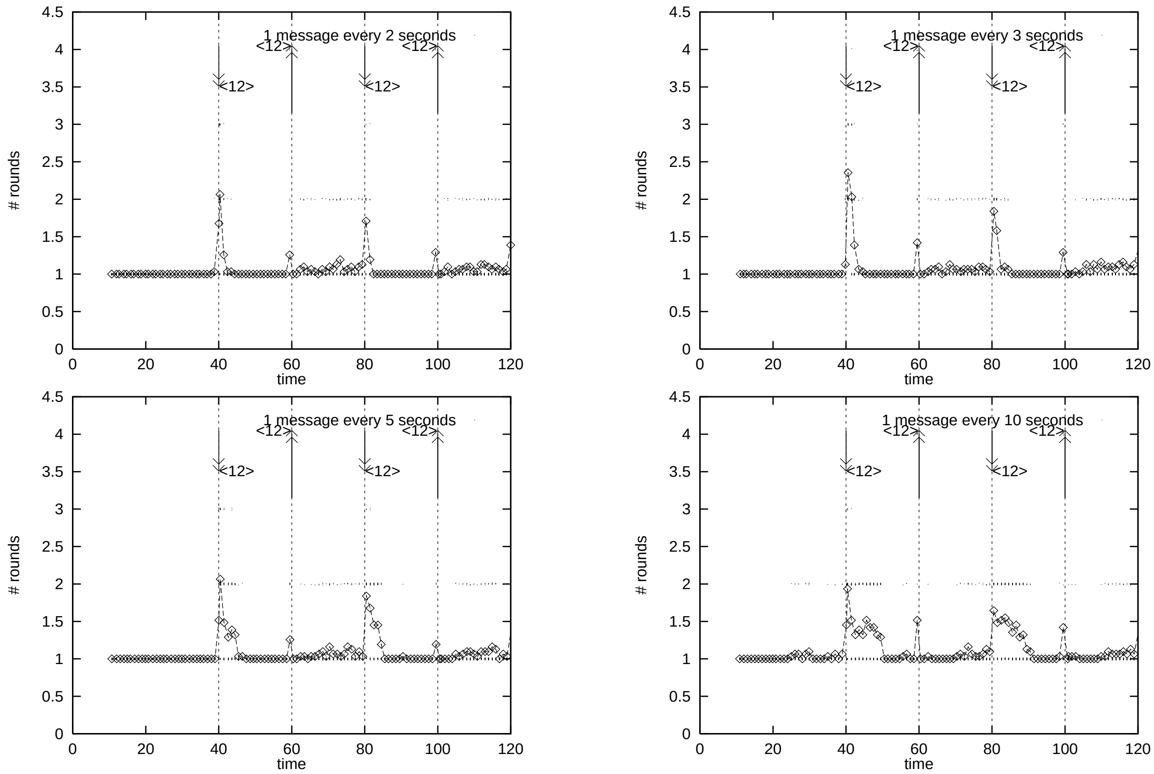


Figure 26: Repair message rounds as a function of the frequency of session messages