Chapter 5

Simulation Results

Having described LMS in the previous chapter, we now proceed to evaluate the performance of LMS and compare it with two other reliable multicast schemes, namely SRM [17] and PGM [47]. Since SRM is a scheme that uses no network support, our intention is not to directly compare the performance of LMS and PGM with SRM. Such a comparison would be unfair. Rather we use our simulations as a means to demonstrate what is achievable with schemes that use network support with the performance achieved by schemes that do not. The final decision of whether we should introduce router assistance in the Internet or employ purely end-to-end mechanisms is, of course, highly complex and is well beyond the scope of this thesis. We merely hope to help the reader understand the trade-offs in making such a decision.

Most of the evaluation was done via simulation. In this chapter we present the details of our evaluation, including the chosen simulator, the metrics, the evaluation methodology and the simulation results. In our simulations, we compare the performance of LMS with SRM (which that comes bundled with the chosen simulator), and PGM, whose simulation was written with help from Sherlia Shi.

The simulations in this chapter aim to explore the trade-offs, performance and scalability of LMS. By reading this chapter, the reader will understand how LMS performs under various topologies and its limitations. The simulator chosen to carry out our evaluation is the UCB/LBNL/VINT Network Simulator - *ns* (version 2)[48]. The topologies used in the simulations were generated via

GT-ITM[49], which is quickly becoming accepted among the Internet community. Several simulation scenarios were used; the topologies generated although not very large (about 200 nodes), were the largest topologies possible, limited by the amount of memory in our machines. Therefore, we have not tested these schemes with extremely large topologies, but we believe that our conclusions still apply to such topologies.

Our simulation results demonstrate the behavior of LMS, SRM and PGM in terms of duplicates and latency. Our results show that both LMS and PGM perform significantly better than SRM, as one might expect, since LMS an PGM have the advantage of network support. For example, recovery latency in LMS is 30 - 60% of the unicast latency, in PGM is close to the unicast latency, but in SRM it is typically higher than twice the unicast latency. The difference between LMS and SRM is a factor of 5-10 times. LMS performs better than PGM by a factor of 1.5 - 3 in terms of recovery latency, because LMS allows receivers to send retransmissions, whereas in PGM only the source is allowed to retransmit. In terms of duplicates, PGM performs the best, but at the expense of maintaining per-packet state at the routers; LMS performs very well without such a penalty, with duplicates limited to only a few percent. SRM without local recovery generates 4-6 duplicates per lost packet, which is quite high and negatively impacts scalability.

This chapter is organized as follows: in the next section we give our reasons for using simulation to carry out our evaluation, followed by the justification of our selection of simulator. We then give our simulation methodology, and our evaluation metrics. We then briefly describe the topology generation tool, namely GT-ITM. Then we describe in detail the simulation parameters, followed by a discussion on simulation verification. Finally, we present the simulation experiments for each of the three schemes, and conclude the chapter with a summary and discussion of the results.

5.1. Why simulation?

The Internet is a very complex entity - one that we do not yet fully understand [50]. Therefore, evaluating schemes such as LMS, which aim to be deployed on the Internet at a global scale, is very hard. In deciding on how to evaluate LMS, we were confronted with the problem that plagues every newly proposed Internet scheme: proper evaluation can be done only after deployment, but large scale deployment is very costly, justified only after the scheme has gone through extensive testing and is relatively mature. Evaluating schemes like LMS that require modifications to the network internals (routers) complicates matters even further, because it alters the existing network behavior, possibly changing some of our current limited understanding of the network.

An alternative to a large scale deployment would be to create an LMS-aware overlay over the current MBONE, (e.g., an LMS-Bone), and use it to evaluate LMS at a smaller scale. Such an overlay could be created using tunnels, just like the MBONE is an overlay over the existing Internet. This is the approach that we expect will be followed if LMS is incrementally deployed. However, adopting such an approach at this early stage poses several difficulties:

- we need access to machines on a large number of sites, preferably distributed over a wide geographical area, which is hard to achieve. Then we need to manually configure these machines.
- even if we could gain access to a large number of machines, deploying LMS requires that we modify their kernels. This poses two problems: network administrators, may not, in general, be eager to allow outsiders kernel access to their machines. Additionally, unless we port LMS to several platforms (a daunting task), we would be limited to machines running NetBSD, the current platform of our LMS implementation. Despite being an excellent research platform, NetBSD is not a very popular operating system. The source code for popular OS' like Windows and Solaris is either not easily available, or not available at all, which greatly limits our choices.
- ideally, we would like access to a site's access router, to avoid manually configuring machines to use artificial routes. However, modifying such routers is a dangerous proposition, since testing may create problems that can seriously affect regular Internet traffic.
- debugging a machine located far away is very difficult. For example, when the remote machine's kernel crashes, it is very hard to fix the problem quickly. Typically, kernel debugging is done by directing the console output of a machine to another via a serial line, which would double the number of machines required (that's assuming we could convince the system administrator to install such serial lines and give us access to more machines).
- even if we managed to overcome the above difficulties, testing LMS on a real network at this early stage is still not a good idea. The reason is that the Internet presents a very

complex environment that we have no control of. Thus, if we observed a certain behavior, it might be impossible to deduce if it is due to LMS or some other factors in the network. LMS must be understood well in the laboratory before being tested in the real network.

For the reasons cited above, we decided to use simulation to evaluate LMS. Choosing simulation, in addition to circumventing the above difficulties, offers several other advantages:

- allows an almost arbitrary number of endpoints and arbitrary topologies. A simulation can be configured to use practically any type of topology, either arbitrary, or one modeled after a real topology.
- simulation provides complete control over the environment. For example, during evaluation we like to control loss (both in terms of number of drops and their location). This is very easy to do with simulation.
- simulation allows sharing of code with others. Thus, results can be duplicated and verified by other researchers, who may also want to evaluate other scenarios, leading into a more complete evaluation.
- with simulation, other schemes can also be simulated using exactly the same parameters (topology, loss, transmission rate, etc.). This allows for a better and more meaningful comparison between schemes.

5.2. Why *ns*?

When it came to choosing a simulator, the choice was relatively easy. Initially, we started by creating our own simulator because we wanted to evaluate LMS in isolation; soon it became apparent, however, that this route would not allow us to leverage off a tremendous amount of support provided by simulators like *ns*.

Ns is a public domain simulator that has gained wide acceptance in the Internet community. It was chosen over other simulators for a number of reasons. Ns contains extensive support for existing Internet protocols, including the TCP/IP protocol suite, unicast and multicast routing, traffic generators, SRM, wireless and LAN support. Thus, *ns* offers a very rich simulation environment. Ns comes with the Network Animator (NAM), which is an invaluable tool in understanding and debugging protocol behavior, and has proven indispensable during the development of LMS. The large *ns* user population encourages code sharing and allows simulations to be shared with other groups. Finally, *ns* is actively being supported, with new modules constantly being added to the distribution. The active support also ensures that bugs are fixed quickly.

Ns, however, has limitations, which the designers are actively working on eliminating. Currently, *ns* is implemented in C++ with a Tcl front-end. The marriage of these two languages, although allowing good flexibility in writing simulations using existing modules, results in a rather steep learning curve and significant complexity when adding new modules. Tcl was chosen to facilitate rapid code development, and as such presents a compromise, trading speed for flexibility. Sometimes this trade-off results in simulations that consume large amounts of memory and consequently, run quite slow. SRM, for example, contains a large portion implemented in Tcl, which severely impacts the size of SRM simulations. In our experiments, we could not run SRM with more than 120-150 nodes on the machines available to us. The LMS and PGM simulations, which are implemented mostly in C++, were much more efficient, both in terms of run-time and memory usage. LMS simulations with 200 nodes would typically finish in a few hours on a 64MB Pentium class machine, whereas SRM simulations would take up to 3 days for 120 nodes on a 1 GB UltraSparc machine.

Despite its limitations, *ns* has served our purposes well. While there is a relatively steep learning curve and much time was expended studying the code and understanding the internals (documentation is sparse but improving), the effort paid off in allowing us great flexibility in our simulations. During our encounter with *ns*, we have also identified some limitations that we believe should be addressed in future releases of *ns*. The most notable example was *ns*' lack of support for router processing. There is currently no way in Ns for routers to extract packets from a stream, process them and then put them back on the stream, as would happen for example in processing IP options. This was an important limitation when implementing LMS, and we had to modify the *ns* internals in our simulations.

5.3. Simulation Methodology

In this section we describe the simulation methodology used to evaluate LMS. The same methodology was used to evaluate SRM and PGM. However, since LMS is the main subject of this study, our efforts were focused primarily on LMS. For example, we refrained from any attempts to modify SRM and PGM to improve performance, and we did not modify their designs to overcome limitations that were discovered (see the repeated retransmissions experiments with PGM, for example) during our evaluation as this is beyond the scope of our work. However, for experiments common to all three schemes, we kept all the relevant simulation parameters (e.g., topology, loss rate, source rate, etc.) identical.

None of the three schemes comes with a fully functional congestion control mechanism. LMS and SRM have no congestion control at all; PGM sketches out a congestion control mechanism in its specification. However, the authors admit that it is still at the research stage and far from being ready for deployment. While multicast congestion control is an orthogonal issue to error control - the problem that LMS, SRM and PGM are primarily trying to address - and is still the subject of on-going research, it has significant impact on the evaluation methodology. Lack of congestion control implies that we should be careful about how we introduce loss in our simulations: for example, it is of little value to induce loss by adding background load to simulate loss caused by congestion, as we expect to happen in the real Internet. But since these schemes would never be unleashed on the Internet without some congestion control mechanism, the simulation results would be of little value.

While the lack of congestion control means that these schemes must be evaluated further once such a mechanism has been added, it does not mean that useful evaluation cannot be carried out at this point. On the contrary, even if congestion control were available, it is highly beneficial to evaluate the error control characteristics of these schemes separately, in order to study the error control behavior in isolation. While there can be significant interaction between error and congestion control that might necessitate later design changes (for example, a slow-reacting congestion control mechanism combined with a fast-acting local recovery might send packets straight into heavy congestion by triggering retransmissions too fast), studying them separately keeps the parameter space smaller and more manageable. Moreover, multicast applications requirements are extremely diverse, implying that is highly unlikely that one error or congestion control scheme will dominate and completely displace all others. Thus, it is very likely that we will see the development of a variety of modules, each suited for a particular class of applications. The modular design means that error and congestion control modules may be paired in various ways, and selecting a good combination requires a clear understanding the workings of each module individually.

In order to isolate the operation of error control, we opted to use artificial packet drops. In other words, drops were induced by deliberately creating loss on the target link. The overall link bandwidth was kept high and the source rate low in order to ensure that any queue buildup at the routers which might cause further packet drops was eliminated. Although our loss module could be configured to drop packets according to any loss model (e.g., random, exponential, bursty, etc.), we concluded that such types of loss would be of little value. The reason is that since we are only concerned with error control, we are not interested in determining results like throughput, which would be affected by the type of loss used; rather we concentrate on evaluating the overhead of recovery in each scheme, *after* a loss has occurred. For the same reasons, we model drops of original packets only; retransmitted packets are never dropped.

Numerous simulation runs were carried out, in order to explore the behavior of LMS in a wide range of topologies and scenarios. Identical scenarios were used where possible with SRM, and PGM, keeping all common simulation variables the same.

5.4. Evaluation Metrics

As mentioned in the previous section, our evaluation is mostly concerned with what happens after a loss is detected. We do not use traditional metrics like throughput because the overall bandwidth seen at each endpoint depends on the aggregate loss, which in turn depends on loss type and location and are very hard to predict. While some studies have been carried out to attempt to characterize loss in the Internet [39, 51], their results have not been conclusive. Moreover, it is not clear that studying the MBONE at its current state, i.e., a sparse overlay, with (frequently) restricted multicast bandwidth, will yield results which will still be valid when multicast becomes commonly used in the Internet.

To avoid making our own (and most likely flawed) assumptions about the future loss characteristics of the MBONE, we limit our study to just two performance metrics that approximately characterize the overhead associated with each of the schemes. This is a similar approach followed by the SRM designers in their evaluation. For consistency, we use the same metrics that have been used in evaluating other reliable multicast schemes [31], but then we add some new metrics. These metrics are, (a) *normalized recovery latency,* (b) *exposure* (for LMS), (c) *requests/repairs per drop* (for SRM), and (d) *repeated retransmissions per drop* (for PGM). Of these metrics, normalized recovery latency and requests/repairs per drop have been used before; exposure and repeated retransmissions are new metrics that we have introduced. We describe these metrics next. The definitions appear in Figure 5.1.

$$\textbf{Normalized Latency: (LMS, SRM, PGM)} \quad \frac{\textbf{Recovery Latency}}{\textbf{RTT to Source}}$$

$$\textbf{Exposure: (LMS)} \quad \frac{\dfrac{\textbf{Total Duplicates}}{\textbf{Number of receivers}}}{\textbf{Total Drops}}$$

$$\textbf{Requests/Repairs: (SRM)} \quad \frac{\textbf{Total Req/Repairs}}{\textbf{Total Drops}}$$

$$\textbf{Repeated Retransmissions: (PGM)} \quad \frac{\textbf{Total Retransmissions}}{\textbf{Total Drops}}$$

Figure 5.1: Simulation Metrics

5.4.1. Normalized Recovery Latency (LMS, SRM, PGM)

Normalized recovery latency is a metric first used by the SRM designers in evaluating the performance of SRM. It is defined as the latency a receiver experiences from the moment it detects a loss until the loss is recovered, divided by that receiver's round-trip time to the sender. Thus, a normalized recovery latency value equal to 1 means that recovery takes exactly one round-trip time; a value greater than 1 means recovery takes longer than the round-trip time, and a value less than one means recovery takes less than one round-trip time. While it may not seem obvious at first glance, recovery can take less than 1 RTT (from a certain receiver's perspective) when a receiver near the source initiates recovery by sending a NACK sooner than a receiver which is further away. Distant receivers benefit in this case, because a retransmission maybe initiated even before distant receivers have detected the loss. All three schemes use gap detection to detect missing packets. However, in

LMS the normalized recovery latency equals the latency from the moment an out of-sequence packet is received, which in LMS is the same as when a NACK is sent, until the packet is recovered, divided by the round-trip time of that receiver to the source. In SRM and PGM, NACKs are not sent immediately upon the detection of a gap, and thus normalized recovery latency includes the back-off time in these schemes.

Another way to look at normalized recovery latency is as a measure of recovery latency compared to the unicast case, where recovery takes at least one RTT. Thus, normalized recovery latency is a measure of how much better (or worse) latency a receiver experiences as a result of using multicast instead of unicast.

Recovery latency is an important measure of the scalability of a reliable multicast scheme. Since all of the described schemes use receiver-reliable semantics (where the receivers, not the sender, are individually responsible for recovery), high recovery latency may result in failure to recover because retransmission buffers were purged. Conversely, low recovery latency will allow the sender (and the receivers) to purge buffers early, resulting in lower resource requirements. Finally, while some applications may be less sensitive to recovery latency (like file transfer), for others latency may be critical (e.g., multimedia applications), thus lower the recovery latency may result in higher application utility.

5.4.2. Exposure (LMS)

The second metric we used in our evaluation is exposure. This metric had not been used in any other study before. It applies to LMS because LMS has local recovery. Exposure is the average number of duplicate messages received by a receiver as a result of loss at some part of the multicast tree which, may or may not have affected the receiver. It is similar to the next metric, duplicate requests/repairs, used by SRM, but it takes into account the presence of local recovery.

Exposure is a general metric, which attempts to capture the number of unwanted messages that reach a receiver as a result of recovery. Recall from earlier chapters that depending on the recovery scheme, a receiver may receive one or more requests or replies, none of which it may need. For example, a scheme like SRM which has no local recovery, will force all receivers to receive all packets generated in response to a loss. Moreover, because of SRM’ s trade-off between latency and duplicates, receivers may actually receive multiple copies of the same request and the same reply.

LMS, due to its local recovery mechanism, only allows one request to escape, which reaches a single receiver outside the loss tree to ask for a retransmission, and therefore creates no duplicate requests *outside the loss subtree*; it allows multiple requests to reach receivers inside the loss tree, but we did not count these as duplicates because these receivers have been affected by loss[1]. On the other hand, LMS does not prevent retransmissions from reaching parts of the tree that do not require them; these we do count as duplicates.

We did not use exposure as a metric to evaluate PGM, because PGM (in most cases) will not allow a receiver to receive a retransmission unless it has specifically requested it. Therefore, PGM does not suffer from exposure.

Like latency, exposure is an important measure of scalability. Schemes with high exposure (like those without local recovery) are less likely to be as scalable as schemes with low or no exposure. Each unwanted message not only wastes network resources, but it may contribute to congestion and incur unnecessary burden at the receivers in the form of interrupts and protocol processing. For sufficiently large groups and non-negligible loss probability, high exposure may result in possibly doubling (or more) of the required bandwidth for the group, since nearly every packet may be lost at some link. High recovery traffic may also adversely impact congestion control since receivers must take into account both original and retransmitted data, in making congestion control decisions.

5.4.3. Duplicate Requests/Repairs (SRM)

Since SRM has no local recovery at the moment (see Chapter 2), we used the same metrics employed by the SRM designers in their evaluation. In addition to normalized recovery latency, we measured the total number of requests and repairs generated for each packet drop. Note that this metric is related to (but not the same as) the exposure metric used for LMS.

5.4.4. Repeated Retransmissions (PGM)

Recall from Chapter 3, that in PGM all retransmissions come from the source (ignoring the presence of DLRs for the moment). Also recall that NACKs set up state as they travel towards the source, and retransmissions erase that state as they travel towards the receivers. In some topologies,

---

1. In a strict sense, these requests constitute overhead, because they result in no useful work for recovery. However, they may be useful in another context (see discussion on other uses of LMS).

it is possible that the retransmission state created by a nearby receiver's NACK may be wiped out by a retransmission even before the NACK from a distant receiver has the opportunity to update the state by marking more links as needing the retransmission. In these cases, the sender is forced to send multiple retransmissions to repair the same loss. In our experiments we measure how often this situation arises and how many retransmissions the source has to send to repair a single loss.

In PGM, the source is already burdened with serving every retransmission request. Repeated retransmissions, if they occur at sufficiently high volume, will increase this burden even more. One way to reduce the problem is with the deployment of DLRs, which increases the resources required by PGM. Alternatively, repeated retransmissions can be reduced if the source delays a retransmission until NACKs are given ample opportunity to set up the appropriate state in the network; This, however, increases recovery latency. How to determine the appropriate value to delay the retransmission is a subject of on-going research by the PGM designers.

5.5. Topology Generation

Topology generation is an important aspect of our simulation experiments because the performance of multicast protocols is often sensitive to the underlying topology [52, 53, 54]. By topology we mean both link topology, i.e., the way links are connected together, as well as receiver topology, i.e., the way receivers are distributed over the multicast tree. Both link and receiver topology affect recovery in a similar manner. Consider, for example, the factors that determine how many receivers are served through a particular link. In wide-area networks connected through a backbone, few links are typically used to connect autonomous segments (ASs) to the backbone, and one AS to another. These links are more likely to carry packets that will be delivered to a large number of receivers, meaning that this link topology may result in many "heavy[1]" links. In a mesh topology, more links may be part of the multicast tree for a given set of receivers, and thus each link may have less receivers downstream, which results in "lighter" links. Receiver distribution affects recovery in a similar manner: tight clustering of receivers will result in more heavy links, whereas even receiver distribution along the multicast tree will result in lighter links.

1. We call them "heavy" in an attempt to capture a feel for the number of receivers "hanging" off them.

To summarize then, topology affects recovery because the location of a packet loss changes the number of affected receivers. Typically, a packet loss near the source results in most receivers not receiving it, but if a packet is lost near the edges, only a few receivers may miss it. In addition, topology may affect recovery in other manners which are protocol specific: for example, schemes that rely on receiver collaboration for recovery may perform better with receivers which are evenly distributed, or have widely varying RTT to the source.

5.5.1. Topology Generator: GT-ITM

The Georgia Tech Internet Topology Models (GT-ITM) [49] was born out of the observation that a large number of simulation studies of algorithms and policies on the Internet have been carried out using topologies generated by models that are far removed from real topologies. Such models included regular topologies (e.g., rings trees and stars), “well known” topologies of existing networks, and random topologies. These three models either offer poor approximations to real topologies, or apply only to specific networks.

GT-ITM attempts to rectify this problem by developing topology construction models that have characteristics which resemble real networks. The result is a topology generator that is both flexible and fast in generating arbitrary size topologies, with characteristics similar to real Internet topologies. Contributions of GT-ITM include the Locality Model, which uses edge length in determining edge probability to construct random graphs, and the Transit-Stub model, which is a hybrid generation method that combines smaller random graphs. The transit-stub model is a structure that closely resembles that of the Internet, and unlike random models, the TS model can generate large graphs while keeping the average node degree low. GT-ITM has been recently incorporated into the *ns* distribution for automated scenario generation.

5.5.2. Selected Topologies

For our simulations, we selected to experiment with three types of topologies: binary trees, random topologies and transit-stub topologies. Random and TS topologies were generated with GT-ITM. The binary tree topology, while an unrealistic topology (i.e., unlikely to be encountered frequently or at a large scale in the Internet), was chosen because it represents a regular, easy to visualize topology. It is also a good topology to test cases with many receivers, as the number of receivers scales exponentially with the height of the tree. Binary trees are a difficult case for both randomized and hierarchical protocols: randomized protocols have difficulty selecting appropriate timer back-off values when the distance of all receivers from the source is approximately the same; hierarchical protocols have difficulties selecting appropriate helpers when all receivers are equally good (or bad) candidates, since they are located at the leafs. Thus, with binary trees both types of protocol are tested under adverse conditions, but with a topology that is easy to visualize.

Random topologies generated with GT-ITM, are representative of the topologies one might encounter in dense (richly connected) parts of the Internet, such as in a large organization or a campus network. We expected (and our results have confirmed it) that random topologies would be a relatively good case for both types (i.e., randomized or hierarchical) of protocol. Transit-stub topologies were also generated with GT-ITM, and are representative of wide-area topologies, composed of a relatively sparsely connected backbone leading into dense clusters. Transit-stub topologies are hierarchical, and we simulated 3 levels of hierarchy in our experiments, reflecting the common backbone-to-ISP-to-local-organization structure. TS topologies are a more difficult case than random topologies, because receivers within a cluster have similar latency from the source, but not as hard as binary trees. Sample topologies are shown in Figures 5.2 and 5.3.

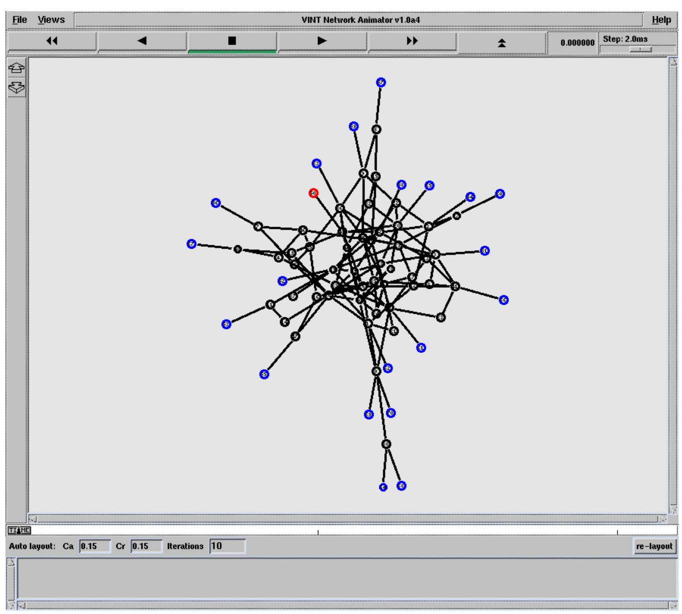

Figure 5.2: Sample random topology

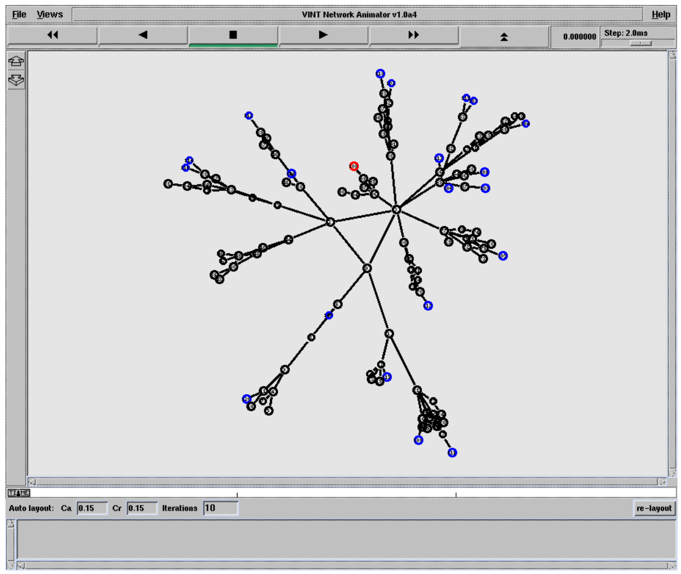

Figure 5.3: Sample Transit Stub topology

While other types of topology may be easily simulated (and we encourage users of our contributed *ns* code to do so), we do not feel that simulating more topologies at this point would have offered any further insight, at least for terrestrial networks. If LMS is to be used in other types of networks (i.e., satellite or wireless networks) then other types of topology may be more appropriate. We have not explored the performance of LMS in these types of networks.

5.6. Simulation Parameters

As described in the previous section, we considered three types of topology in our simulations, namely binary trees, random and transit-stub. The link bandwidth for all links was set to 1.5 Mbps, and the source rate at 10 packets/second, with packet size ranging from 240 to 1024 bytes. Note, however, that the actual values of these parameters are not important as long as they are selected such that no congestion is observed on any link. The reason is that we are interested in studying what happens after a loss occurs (see previous discussion). The link delays were set by GT-ITM at topology generation.

For binary trees we simulated trees of height ranging from 3 to 7 (8 to 128 receivers). For random and transit-stub simulations, we used topologies containing up to 200 nodes (100 internal nodes and 100 receivers). We generated 10 random and 10 transit-stub topologies, each containing 100 nodes. We run simulations with 5, 20 and 100 receivers, randomly distributed on the internal nodes. For each topology we ran 10 simulations, each time with a different receiver allocation (generated by feeding a random seed to the random number generator). Thus, each presented plot is the result of 100 simulation runs. For random topologies the receiver placement was random on all internal nodes; for transit-stub topologies receiver placement was also random, but restricted to stub nodes.

As discussed earlier, the loss characteristics of the current MBONE are still unknown. Thus, we did not attempt to make any assumptions about the loss characteristics (e.g., loss duration and location); instead, we chose to collect results using a single packet loss; similarly for loss location, (which has significant impact on performance), we chose to investigate the following three cases:

- **Loss at the source:** with this type of loss a packet is lost on the link from the source to the network so that none of the receivers receives the lost packet. This case tests how a scheme responds to widespread loss, e.g., how it avoids NACK implosion.
- **Loss at each receiver:** here loss affects only a single receiver. We ran a simulation with loss on one receiver; then, we moved the loss to the next receiver and ran the simulation again, until all receivers were covered. This is the opposite of the previous case, and tests how the scheme reacts to localized loss, which affects only a small portion of the receivers.
- **Loss at each link:** this case attempts to cover the ground left unexplored by the previous two. Loss is moved from link to link with an entire simulation run each time, until all links are covered. This is equivalent to random loss where all links have equal loss probability.

While we certainly do not claim that these cases represent real loss characteristics of the MBONE, we believe that they provide sufficient information to attain a basic understanding of the behavior of the error control schemes under a variety of loss patterns. As we have pointed out in the previous chapters, for all schemes there are pathological topology/loss combinations that lead to severe degradation in performance. It is too early to tell if any of the pathological cases will actually appear in real networks. As we learn more about the loss characteristics of the MBONE, updated loss models can be plugged in our simulations to obtain more precise results.

5.7. Simulation Verification

Our simulations are composed of two separate components, namely the forwarding services offered by LMS, and the reliable multicast protocol. We verified the correct operation of these components using two separate methods: (a) with simulation traces, and (b) by visualizing the packet flow using the *Network Animator (nam)* [55].

5.7.1. Verification using traces

Our simulations contain a substantial amount of debugging code which can be turned on or off with a flag. When the debugging code is turned on, a detailed packet trace is generated as the packets flow between the sender, routers and receivers. We have generated numerous such traces and manually verified the correct operation of both the forwarding services and the error control mechanisms.

5.7.2. Verification using Nam

Nam is a companion tool to *ns*, which allows visualization of traces generated by *ns*. Nam provides operations like play, pause, fast-forward, rewind and reverse playback of *ns* traces. We have found nam to be an extremely valuable tool in debugging our simulations, and we have used it almost religiously. Nam was well suited for debugging the forwarding services because it allowed us to see at a glance if a packet was either misforwarded or not generated all. Similarly, it was invaluable in debugging our error control mechanism because of its deterministic nature. We have created numerous nam animations during the course of the development of our simulation and spent a substantial amount of time ensuring that the animations portrayed the expected behavior.

5.8. LMS Experiments

We have simulated LMS in a variety of topologies and various session sizes. We only simulated a single source sending data to many receivers (also referred to as a one-to-many communication). We do not expect that results would change with multiple sources.

It is very important to note that unless otherwise stated, in the LMS experiments the repliers were kept static. As we discussed earlier in Chapter 4, the reason is twofold: (a) without knowledge of the loss characteristics of the MBONE, it is hard to devise an efficient replier adaptation scheme, and (b) we wanted to explore the performance of LMS with simple replier allocation. With static repliers, the results presented for LMS are not optimal. This is especially true for exposure, where the replier selection is the most important factor governing performance. As our experiments will show, LMS performs very well even with static repliers.

5.8.1. Binary Trees

In the first experiment we simulate LMS with binary trees ranging in height from 3 (8 receivers)

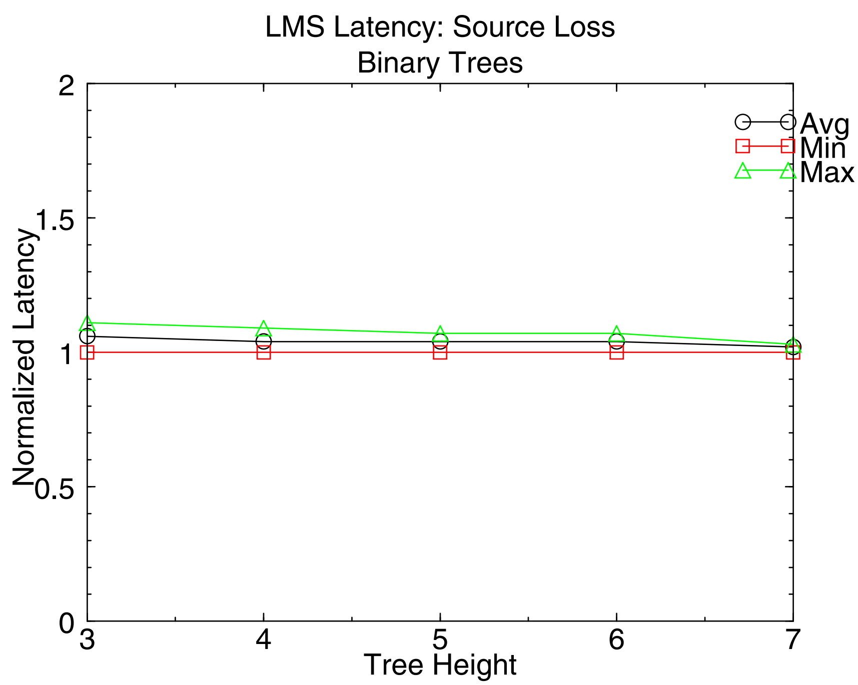

Figure 5.4: LMS latency, binary trees, loss at source

to 7 (128 receivers) and loss at the source. The results are shown in Figure 5.4. We plot the average, minimum and maximum recovery latency. The x-axis lists the five different topologies used in the experiment and the y-axis the normalized recovery latency. We observe that since all receivers have the same RTT to the source, the recovery latency is close to 1. The minor deviations are caused by slight queueing time due to synchronized requests. Note that the recovery latency does not change much as the tree height is increased.

In the next experiment we move the loss to the receivers. The results are shown in Figure 5.5.

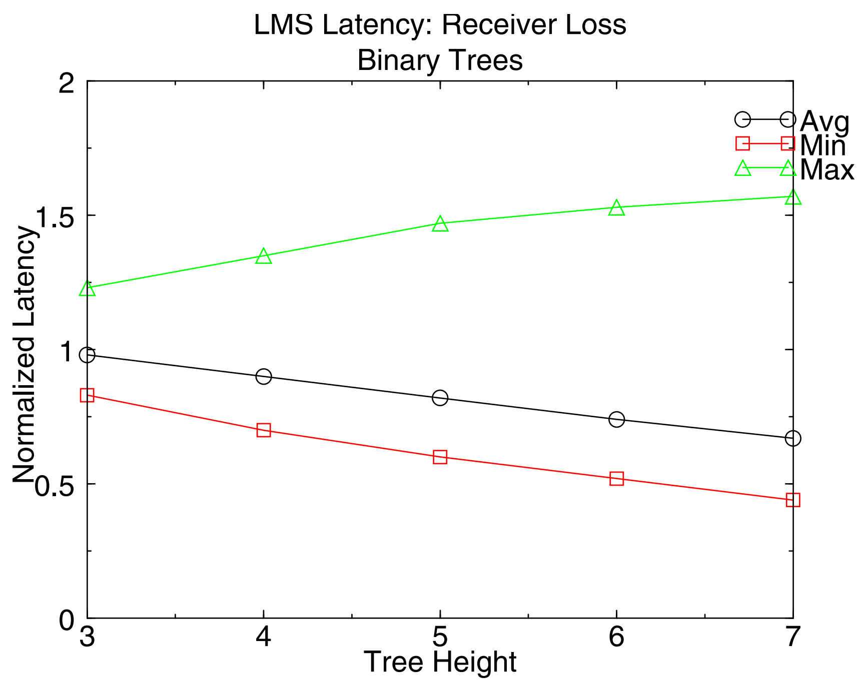

Figure 5.5: LMS latency, binary trees, loss at receivers

Here we observe that the average recovery latency decreases as the tree height increases, so for larger trees recovery is faster on average. The maximum latency increases, because loss at some receivers causes a NACK to propagate back towards the source only to be turned around and delivered to another receiver instead. This causes a loss to be recovered from a receiver that happens to be further away from the requesting receiver than the source, thus increasing the recovery latency to a value beyond 1. Note, however, that in binary trees the recovery latency can never exceed 2. The reason is as follows: the maximum distance (in number of hops) between any two receivers is given by the expression: $d = 2 \cdot (h-1)$. As the tree height increases, the maximum normalized latency becomes:

$$\lim_{h \to \infty} \frac{2 \cdot (h-1)}{h} = 2$$

In the final experiment to measure recovery latency, the loss is moved around from link to link

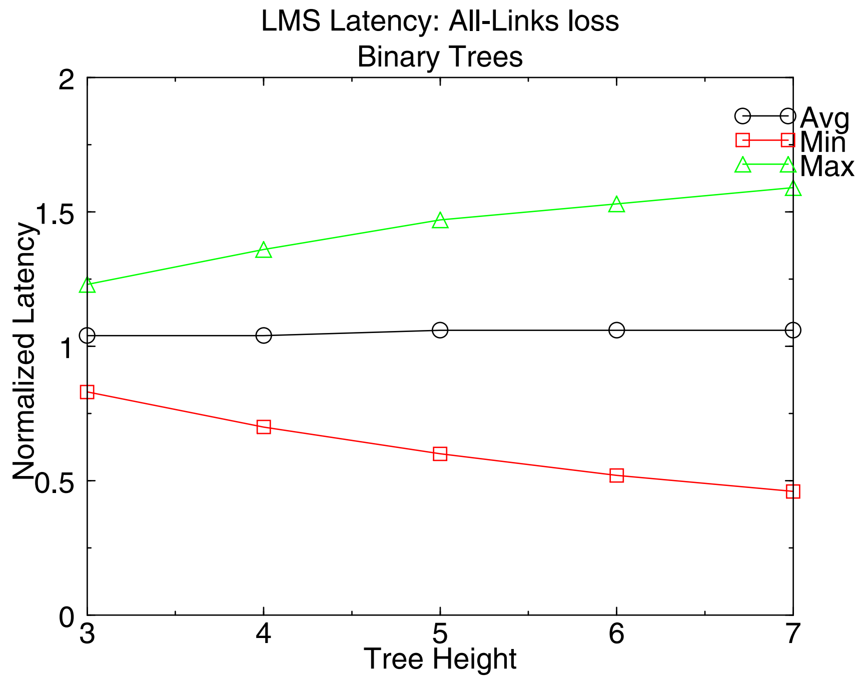

Figure 5.6: LMS latency, binary trees, loss at all links

until all links are visited. The results are shown in Figure 5.6. As the tree height increases, we observe that the average recovery latency remains unaffected and close to 1. The maximum latency increases, but for the reasons described earlier, it will never exceed the value of 2. The minimum latency decreases, because as the tree gets taller the minimum distance between two receivers remains the same while their distance to the source increases.

In the last LMS experiment with binary topologies, we measure the exposure as the tree height increases. Recall that in these experiments repliers are kept static. The results are shown in Figure 5.7. We measured exposure for all loss cases, namely source, receivers and links. Note that the exposure when loss is at the source is zero because all receivers lost the original packet and thus need the resulting retransmission. For the remaining cases, exposure not only starts out low (less than 15%), but it decreases quickly as the size of the tree increases, despite the fact that repliers are static.

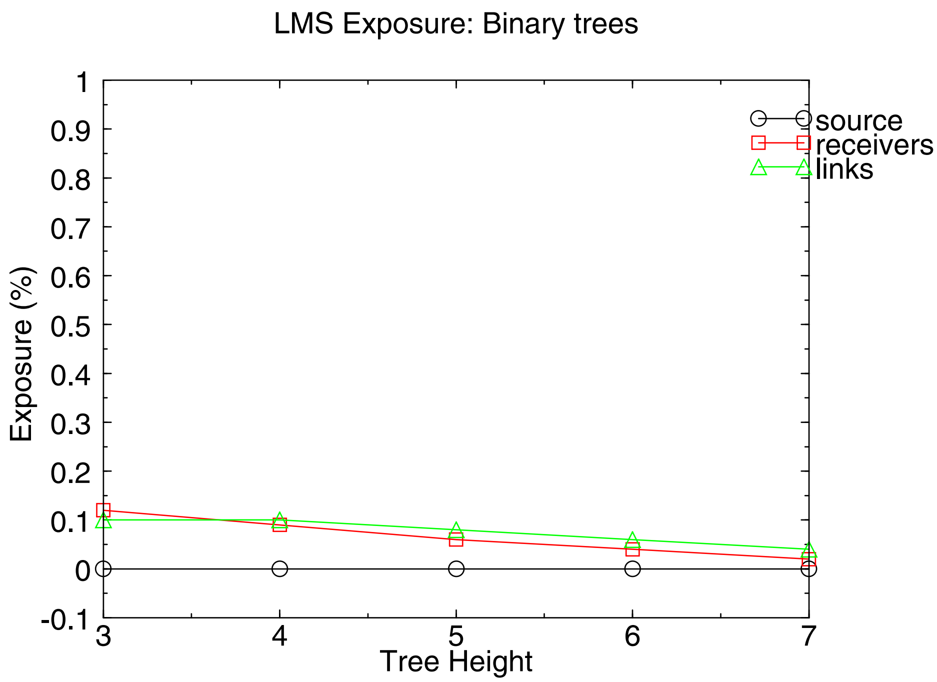

Figure 5.7: LMS exposure, binary trees, loss at all links

In summary, even though binary trees are a difficult topology for LMS because there are no helpers in the internal nodes to speed up recovery and reduce exposure, LMS still manages to perform well. It recovers losses at about one RTT, and keeps exposure very low.

5.8.2. Random Topologies

As mentioned earlier, the random topologies we used in our experiments were generated with GT-ITM. The edges in the topologies were assigned using a random distribution with probability 0.1. We generated topologies consisting of 100 nodes. Then, we randomly assigned 100 receivers and a source to these nodes, bringing the total number of nodes to 201. Each topology was used for 10 runs, each time removing and re-assigning the receivers to internal nodes. We used a total of 10 topologies, which resulted in 100 simulation runs for each experiment. The results are next.

As with binary trees, we start with experiments measuring recovery latency. The first set of results is with loss at the source, and are shown in Figure 5.8. In the figure the term R100-100 means "random graph with 100 nodes and 100 receivers." From the figure we see that on the average, recovery takes about 30-40% of the unicast RTT, which is significantly better than with binary tree

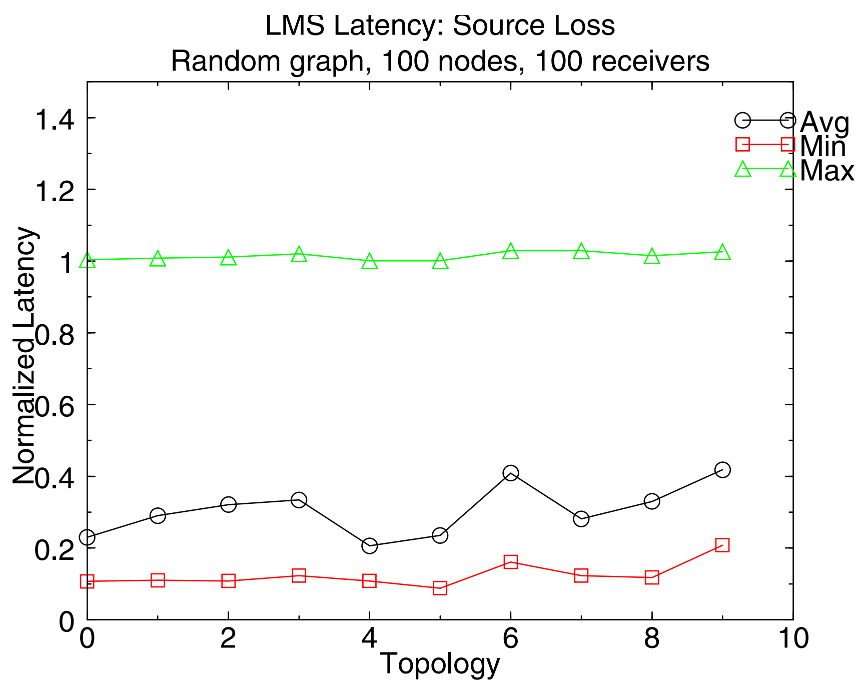

Figure 5.8: LMS Latency, Random topologies, loss at source

topologies. The reason is that in random topologies there are many helpers in the internal nodes, and thus recovery is initiated faster. The maximum latency is again around 1, and is experienced by receivers that are close to the source (note, however, that in absolute terms these receivers recover in less time than distant receivers). The average value is close to the minimum value, which implies that the latency distribution has a long tail.

Our next set of results, shown in Figure 5.9, shows recovery latency with loss at the receivers. Here we observe that the average latency has increased slightly to about 50%. The reason can be deduced by looking at maximum latency, which has increased slightly, to about 1.25. The increase is due to LMS selecting repliers that are located at larger distance that the source, as described in the previous chapter. With loss at all links, as shown in Figure 5.10, the results do not change significantly. The average latency again increases slightly to about 60%. Minimum and maximum latencies are essentially unchanged.

In Figure 5.11, we show exposure for different loss types. The results show that exposure is very low, under 2% in most cases, and well below 3%. For loss at the source, exposure is of course zero. The highest exposure occurs when loss is at the receivers, which is as expected. When loss occurs

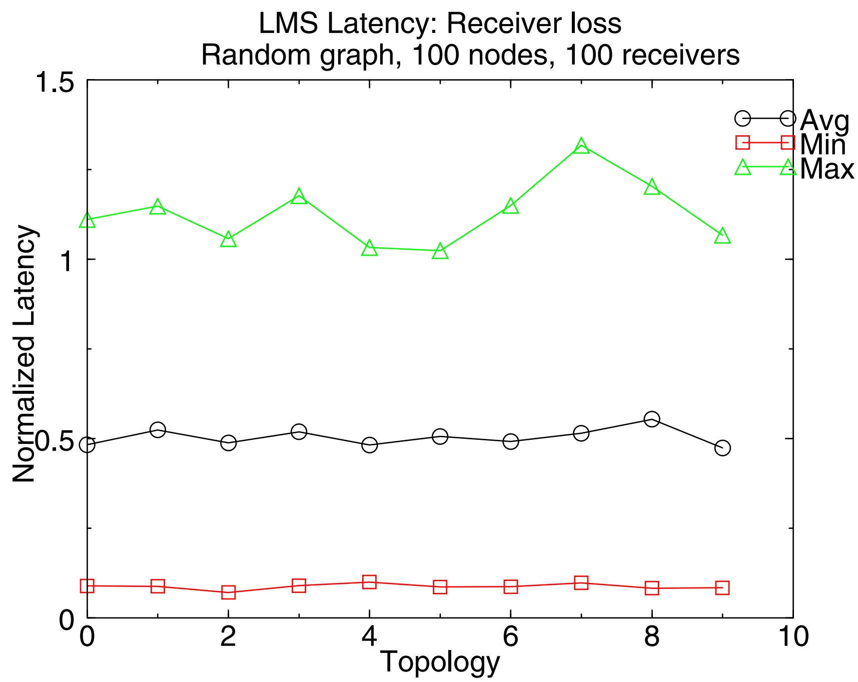

Figure 5.9: LMS latency, random topologies, loss at receivers

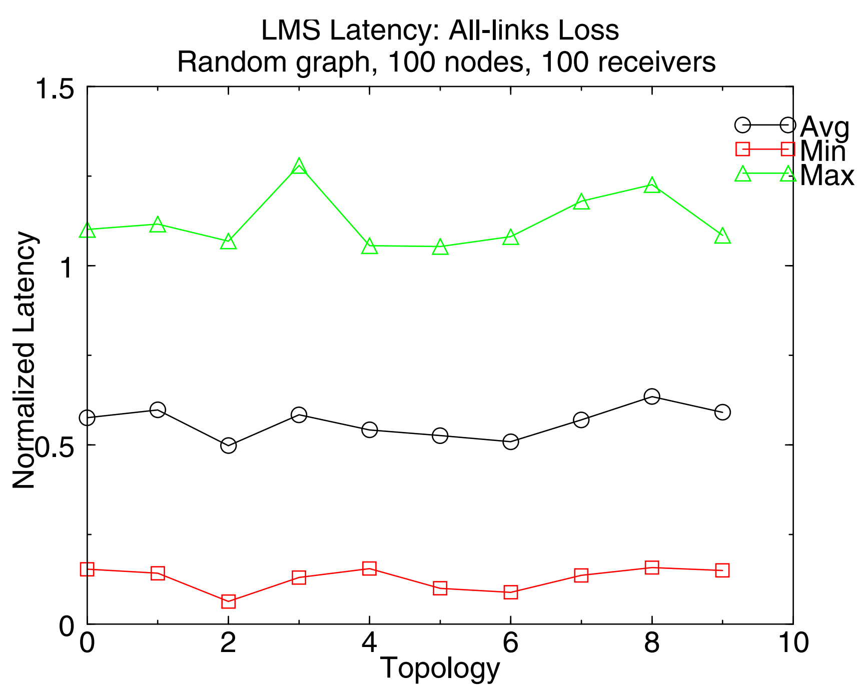

Figure 5.10: LMS latency, random topologies, loss at all links

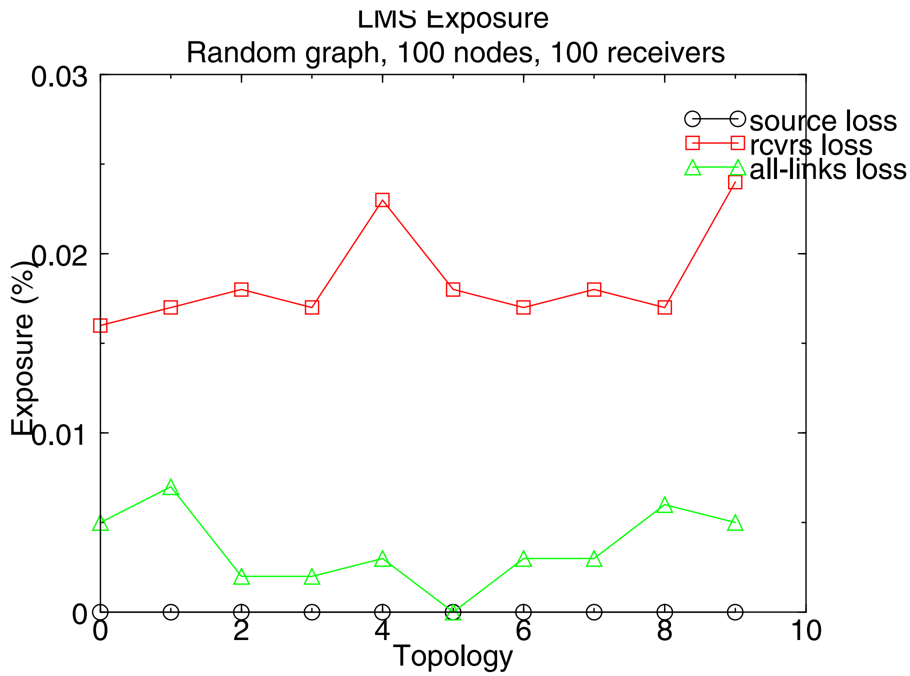

Figure 5.11: LMS exposure, random topologies

at a receiver acting as a replier, recovery causes duplicates to other receivers if they happen to lie downstream of the replier. When loss is equally distributed at the links, exposure remains very low, at under 1%. As with latency, exposure is much better with random graphs than with binary trees, again confirming our initial claim that binary trees are a difficult topology for LMS.

Varying the number of receivers

In the previous experiments, we have kept the number of receivers high to test the scalability of LMS, and have shown that LMS scales very well in large topologies and with large receiver populations. In this section, we present experiments that test the performance of LMS with small, sparse groups. We used the same topologies as in the previous experiments, altering only the number of receivers.

The results are shown in Figure 5.12, and Figure 5.13. We plot simulation results with 5, 20 and 100 receivers, the latter being taken from the previous experiments. We used loss at all links and plot only the average latency. From the results we notice that the recovery latency is inversely proportional to the number of receivers. By definition, recovery latency is 1 when there is only one

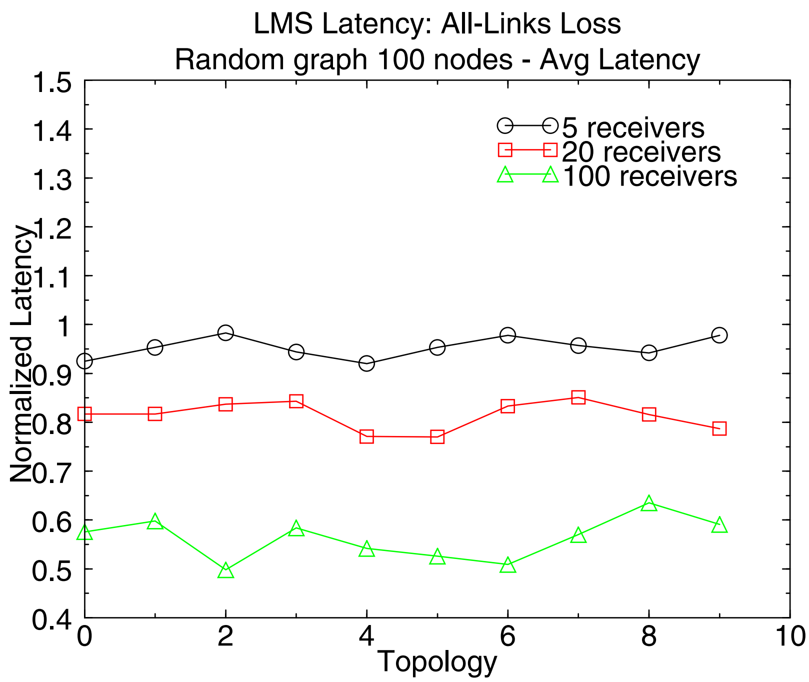

Figure 5.12: LMS latency, random graphs, different receiver population

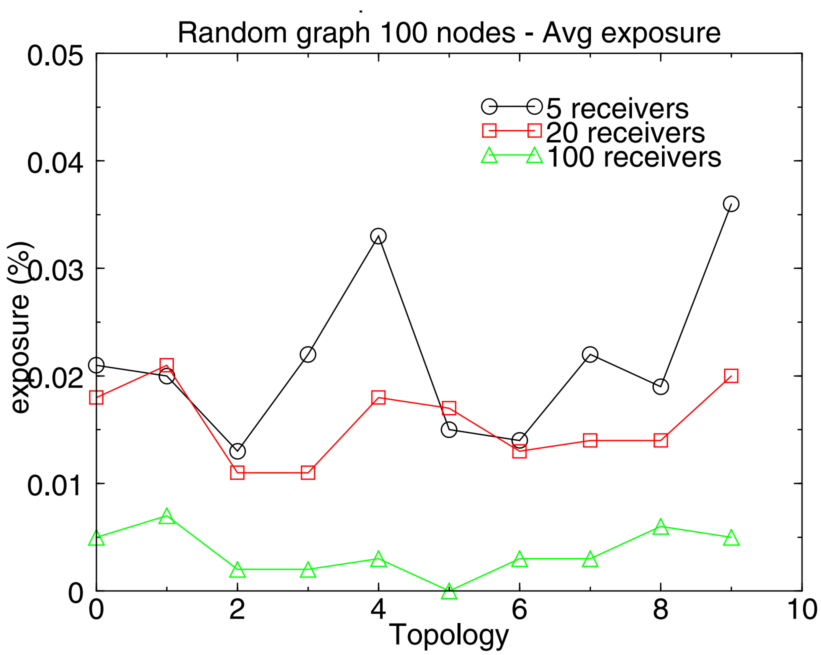

Figure 5.13: LMS exposure, random graphs, different receiver population

receiver, and gets progressively smaller as more receivers join the group. The same conclusion applies to exposure: larger receiver population leads to less exposure. Note, however, that even with few receivers exposure is still very low (less than 4%).

The fact that performance improves as the multicast group gets larger is good news. The reason performance improves with more receivers is that more helpers are available to initiate and send retransmissions, which improves latency; more helpers also means that there is a better chance to find a helper better located to serve retransmissions without causing exposure. The observation that performance improves with larger groups is an interesting result, which is actually not limited to LMS, but also applies to SRM and PGM, in varying degrees. For example, while in SRM latency improves as the number of receivers increases, in order to reduce duplicates from the higher receiver population the timer back-off values may have to be further increased, which can offset the latency gains. In PGM, the presence of more receivers will initiate retransmissions faster, but may worsen the repeated retransmissions problem. Recovery latency is the sum of the NACK propagation latency to the source plus the propagation latency of the retransmission to the receivers. The presence of more receivers may shorten the former, but not the latter. Also since PGM recovers from the source in most cases, the latency reduction will not be as great as with LMS.

5.8.3. Transit-Stub Topologies

In this section, we examine the performance of LMS with transit-stub topologies. While random topologies are a good approximation of dense, richly connected topologies, transit-stub topologies are a better approximation of the hierarchical structure of the Internet.

As with random topologies, we generated 10 topologies with 100 nodes each. The parameters fed to GT-ITM to generate the topologies are as follows: 1 top-level domain (the transit domain) with 4 transit nodes; each transit node had 3 transit-stub nodes; and each transit-stub node had 8 stub nodes. This brings the total number of nodes to $1 \times 4 \times (1 + 3 \times 8) = 100$ nodes. As with random topologies, we randomly assigned 100 receivers, but in this case the receivers were assigned to stub nodes only. Unlike random topologies, instead of simulating loss at the source and receivers, we opted to simulate loss at the different types of links. Thus, we studied loss at transit-transit, transit-stub and stub-stub links. The motivation was that we were interested in finding out how loss at each type of link affects performance. However, as with random topologies, we also produced results with loss at all links.

The results are shown in Figures 5.14, 5.15, 5.16, 5.17, and 5.18. In all figures, ts100-100 means

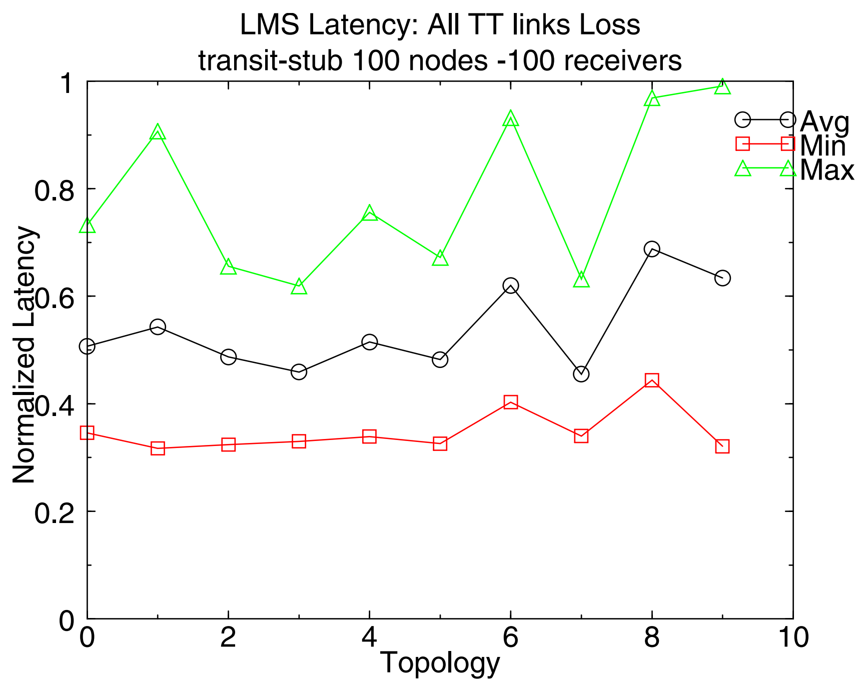

Figure 5.14: LMS latency, transit-stub topologies, loss at transit-transit links

a transit-stub graph with 100 nodes and 100 receivers. While latency remains at about 50% on the average, when all links are lossy, latency is a bit higher for loss at the higher levels (transit-transit links) and less so at the middle level (transit-stub links). However, the difference is small, and in general the results are on par with random topologies. The situation, however, is different with exposure. While exposure remains low with loss near the receivers (on stub-stub links), it increases significantly with loss at the higher levels (transit-transit and transit-stub), reaching peaks of 15-20%. While this is not alarmingly high, it seems to be highly dependent on topology and receiver allocation (for example topologies 0, 4, 7 and 9 have very low exposure, whereas topologies 5, 6, and 8 have somewhat higher exposure).

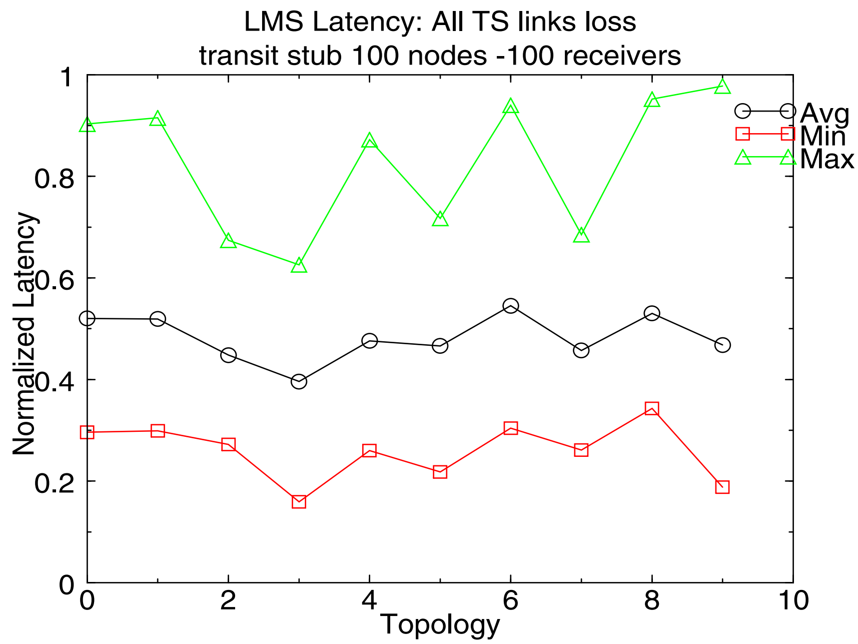

Figure 5.15: LMS latency, transit-stub topologies, loss at transit-stub links

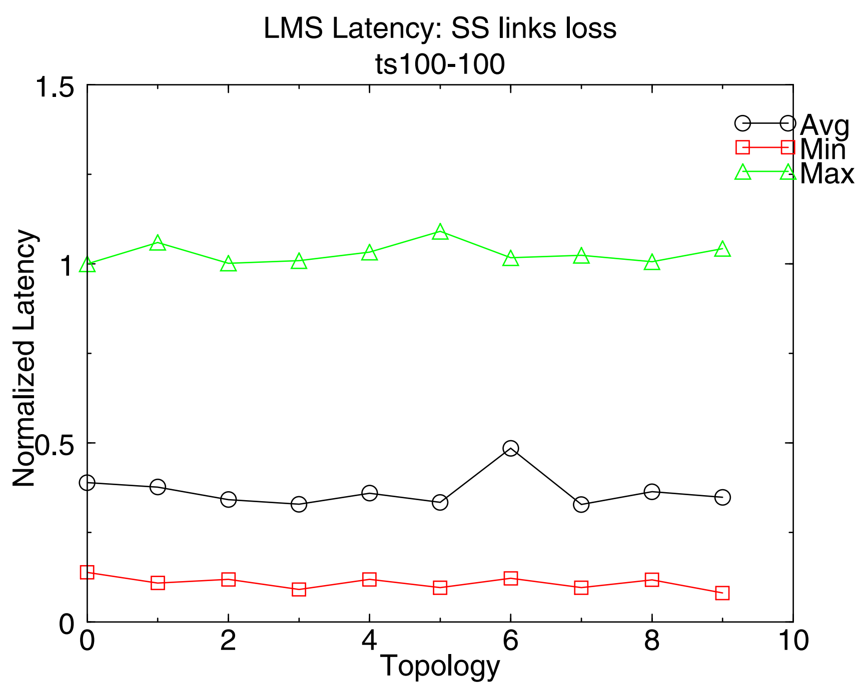

Figure 5.16: LMS latency, transit-stub topologies, loss at stub-stub links

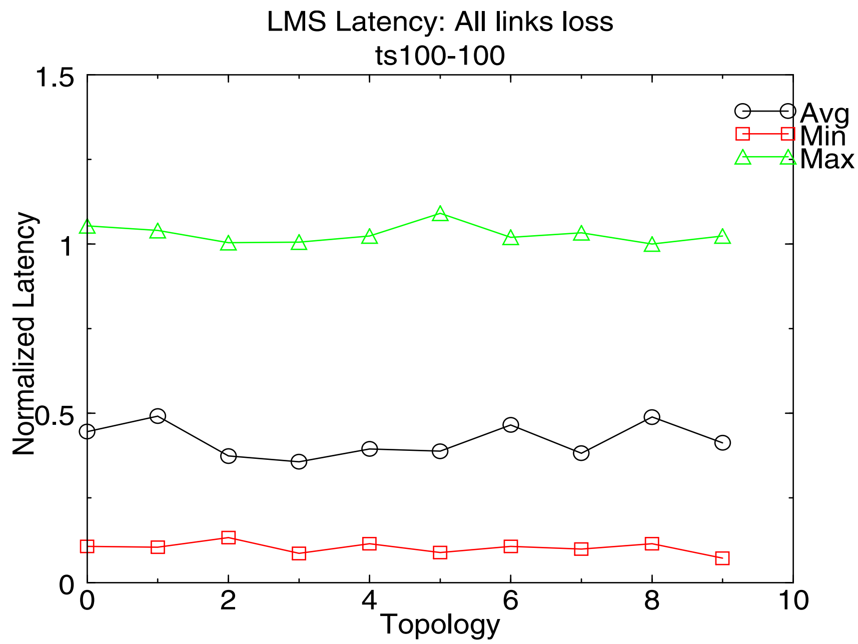

Figure 5.17: LMS latency, transit-stub topologies, loss at all links

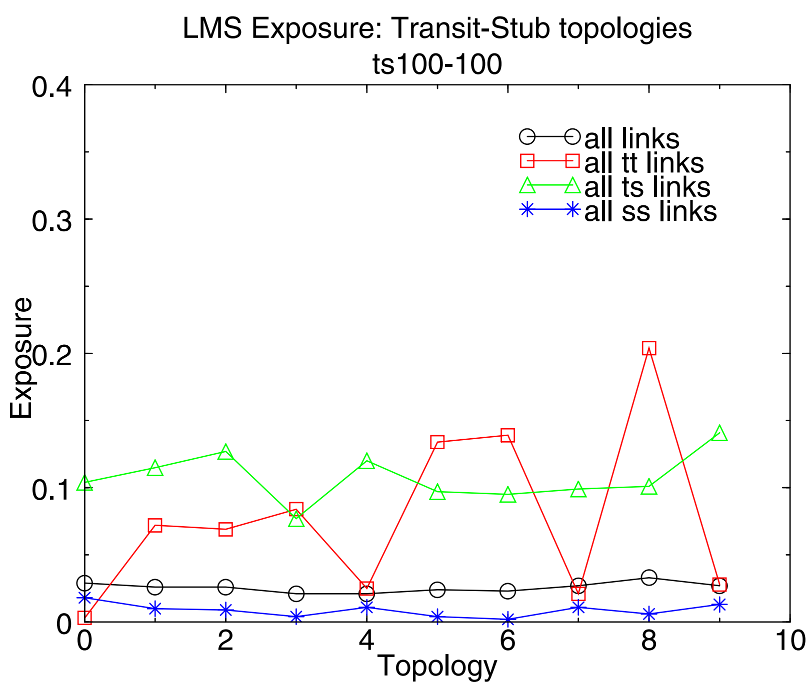

Figure 5.18: LMS exposure, transit-stub topologies

5.8.4. LMS Trade-offs

Having examined the performance of LMS via simulation, let us now summarize its trade-offs. The most important issue in LMS is the proper selection and maintenance of the replier state. Proper selection of repliers will minimize exposure and recovery latency. Frequent replier refresh will guard against replier failure.

Proper replier selection affects exposure because it controls duplicates. With optimal replier selection LMS can eliminate duplicates completely. However, making an optimal replier selection requires that we predict the location of loss, which we assumed we could not do. Therefore, LMS will always incur some amount of exposure as long as the location of loss is unpredictable.

Replier selection also affects latency. Some of our experiments revealed that recovery latency can exceed 1.0. This happens when LMS chooses a replier whose RTT from the turning point is higher that the RTT of the source from the turning point. In the previous chapter we described methods to deal with this scenario by considering repliers from upstream, but at the expense of risking implosion.

The problems above, as well as replier robustness, can be solved by frequent replier state refreshing. However, this may incur a large amount of control traffic, which aggregated over multiple sources and over multiple multicast groups may be unacceptable. We propose to allow receivers to trigger updates when conditions have changed sufficiently (e.g., exposure or latency have become excessive). Routers may simply employ a filter to allow a limited frequency of updates, to guard themselves from malfunctioning receivers.

5.9. SRM Experiments

SRM has already been extensively studied via simulation and results have been reported elsewhere [31]. Our goal here was not to repeat or extend already published results, but to run simulations of SRM on the same topologies used for LMS, thus eliminating one important variable in comparing the two schemes. However, we were not completely successful in achieve that goal. While we used the same topologies for LMS and SRM, we could not run SRM simulations with more than 20 receivers in the 100-node topologies. Attempting to use more receivers resulted in extremely long simulation runs and very high memory consumption. We ended up running the SRM simulations on an UltraSparc with dual processors and over 1GB of RAM, which still took about 3 days to finish each set of 100 runs. The reason the SRM simulations are so slow is that the SRM implementation in *ns* is done mostly in Tcl. Clearly the goal is to maximize flexibility; unfortunately this comes at the price of speed and memory consumption.

Since our goal was to use the *ns* implementation of SRM "as is" and not to study SRM in depth, we decided not to attempt to improve the simulation running time (which would require re-writing the simulation in C++). The results we report are consistent with results presented previously, and thus we do not feel we would have contributed further to the understanding of SRM had we simulated larger topologies.

In the following sections we present results from simulating SRM in random topologies only. As mentioned before, further study of SRM is beyond the scope of this work. The random topologies are the same used with LMS, but here we use only 20 receivers for SRM, to keep memory usage and simulation running time manageable. We report results for normalized latency and the number or requests and replies generated for each lost packet. Recall that without local recovery, these messages will reach every member of the multicast group.

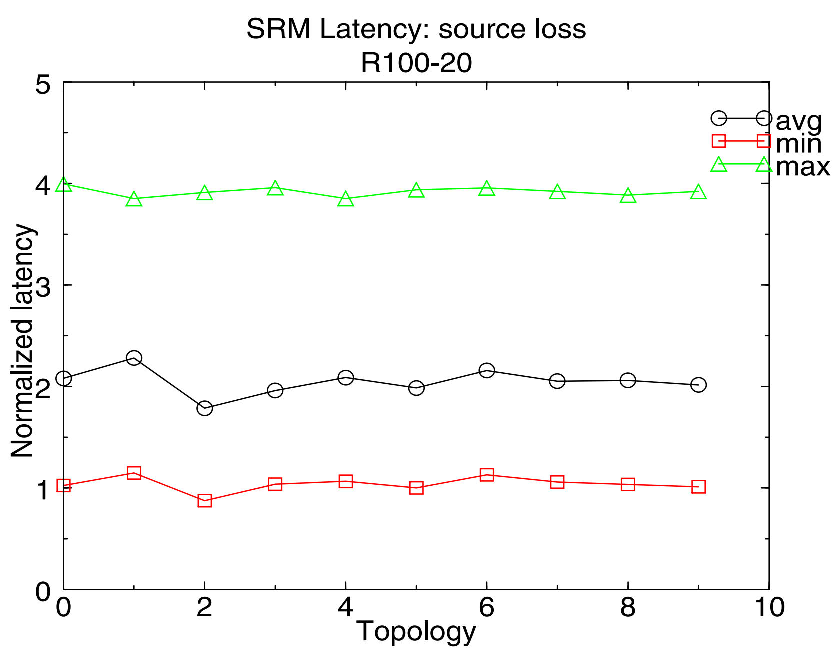

Figure 5.19: SRM recovery latency, random topologies, loss at the source

Figures 5.19, 5.20, and 5.21, show the recovery latency for loss at the source, receivers and links

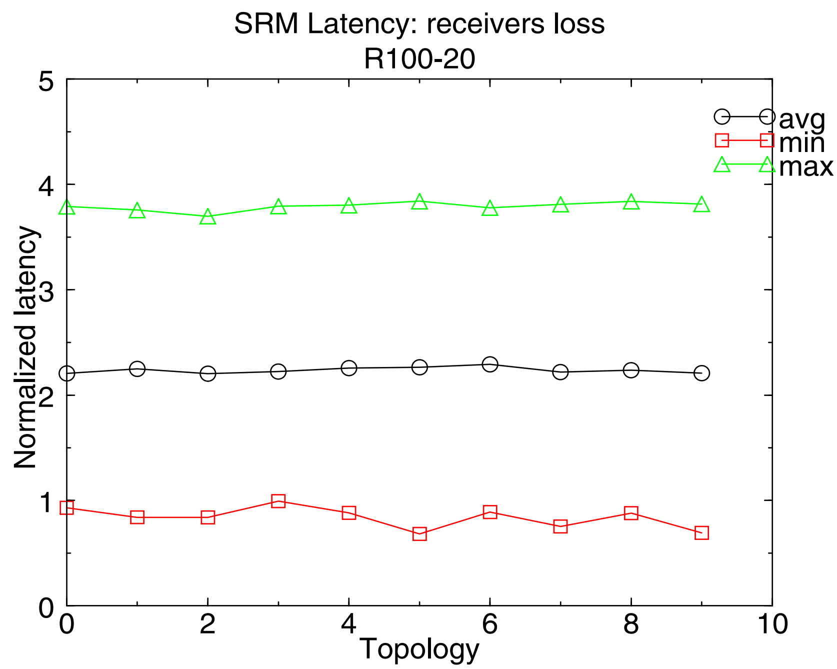

Figure 5.20: SRM recovery latency, random topologies, loss at all receivers

respectively. We note that on the average, SRM recovers from a loss in about 2 RTTs, or twice the unicast latency, with the maximum value being around 4 and the minimum around 1. We also note that the recovery latency is relatively uniform over all topologies. We believe that the reason is that the back-off timers absorb any differences that may arise due to a particular topology. In addition to being relatively insensitive to topology variations, SRM appears to also be insensitive to loss location. Thus, there are very small differences between results from loss at the source, receivers or links.

Figures 5.22, and 5.23, show the number of requests and replies generated on the average for each loss. For requests, the linear component in the back-off timers works well and keeps the number of requests low (a little above 1 for loss on all links). The number of requests are a bit higher when loss is at the source because all receivers compete in sending a request. The results, however, are significantly worse for replies where SRM may generate 4 -5 replies for each loss. For replies

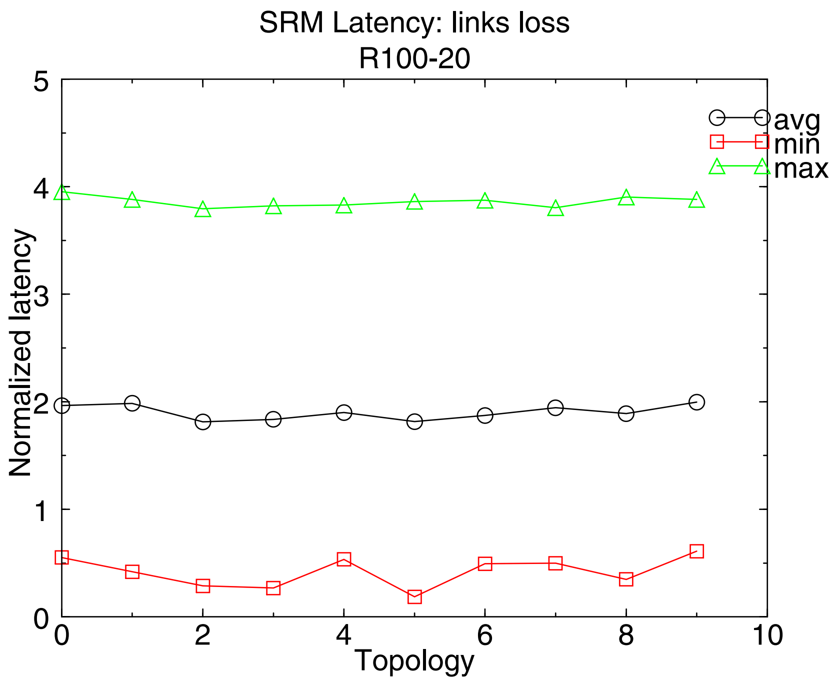

Figure 5.21: SRM recovery latency, random topologies, loss at all links

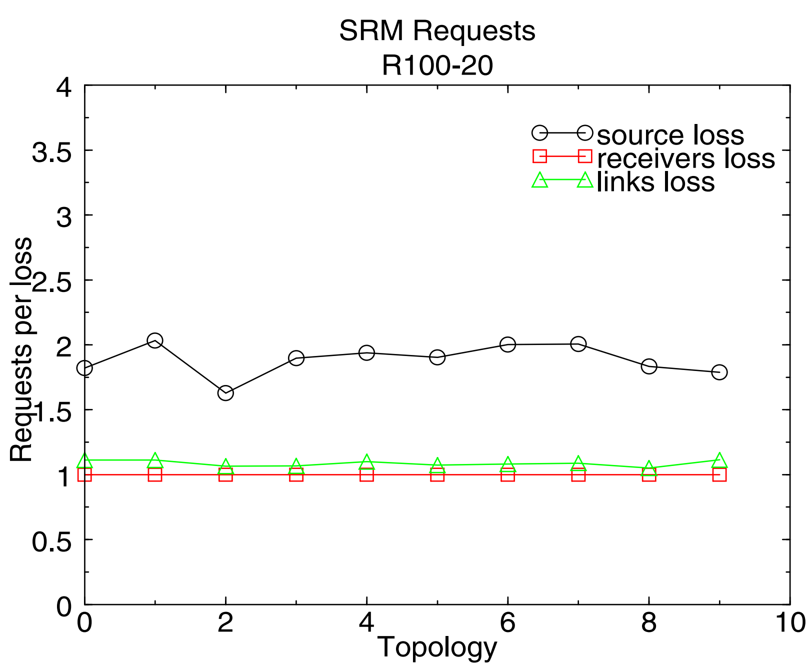

Figure 5.22: SRM requests, random topologies

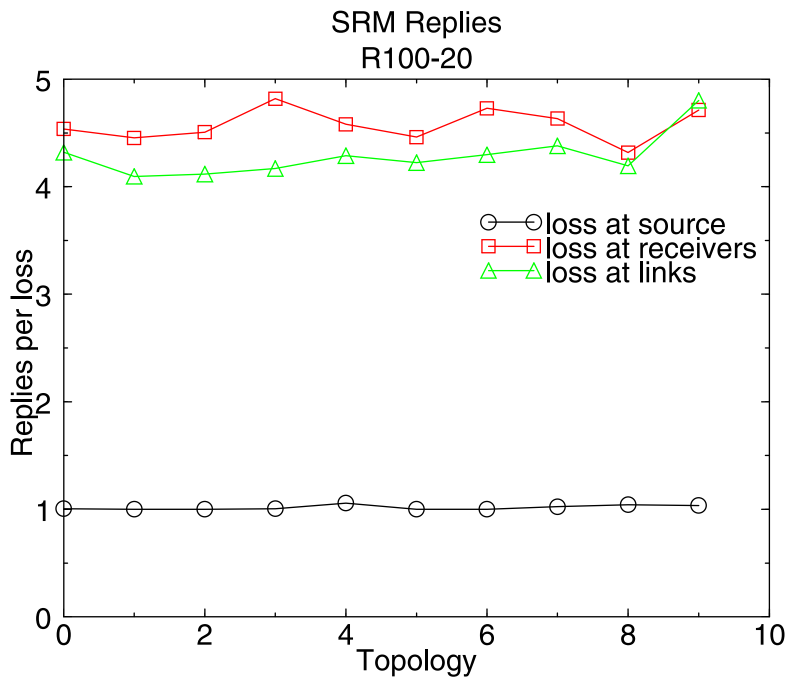

Figure 5.23: SRM replies, random topologies

the linear component is less effective since the inter-receiver RTT is smaller in general than each receiver's RTT to the source.

5.9.1. SRM with Adaptive Timers

As described in Chapter 3, for loss patterns that do not change frequently over time, SRM can make use of an algorithm to adapt the back-off timers and improve recovery latency while minimizing duplicate control messages. The algorithm was proposed by the SRM designers and aims to optimize the back-off timers based on the underlying topology. The SRM implementation in *ns* includes adaptive SRM, and thus we carried out some experiments with adaptive timers. The results are shown in Figures 5.24, and 5.25.

From the results we can see that adaptive timers have significantly reduced the number of duplicate replies in SRM. Without adaptive timers, there would be over 4 replies per loss, but with adaptive timers that number is reduced to less than 2. We do not see a significant improvement in the number of duplicate requests, but the number of such duplicates was low to begin with and would be hard to improve on it.

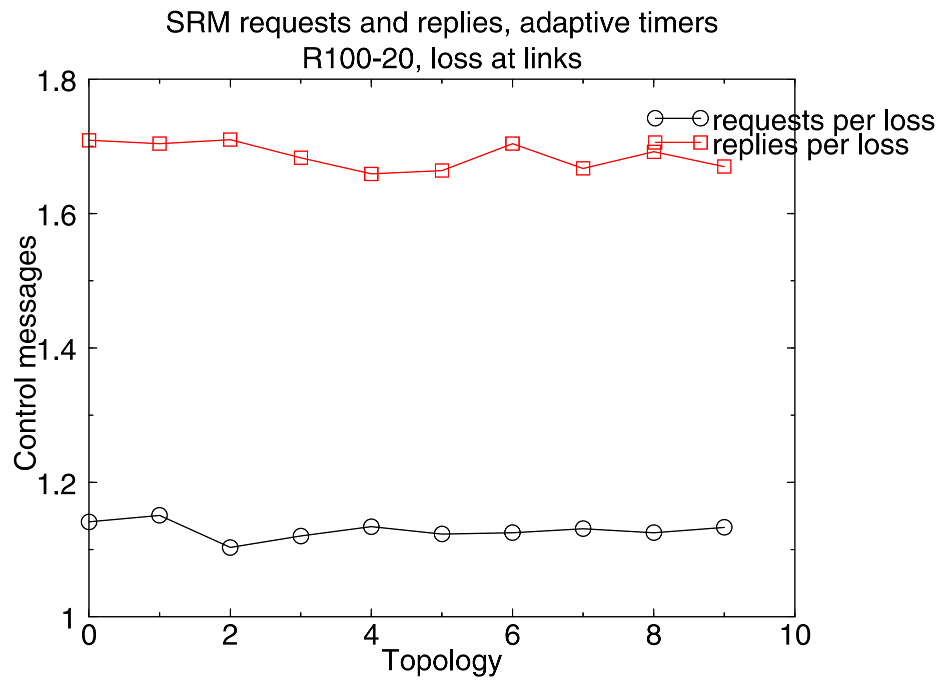

Figure 5.24: SRM requests and replies, adaptive timers

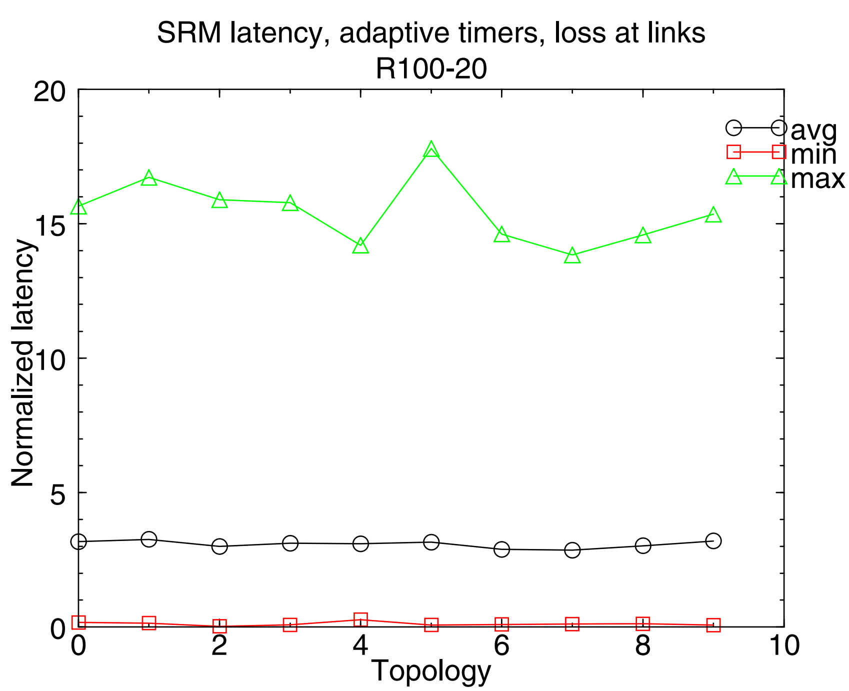

Figure 5.25: SRM Latency, adaptive timers

The penalty paid for reduced number of requests and replies can be seen in the latency results. The average recovery latency has gone up from about 2 without adaptive timers to about 3. The reason can be seen by looking at the maximum latency, which hovers around 15. We believe that the reason for these extremely high latencies, is the high timer values the algorithm assigned to some receivers when loss was at a particular location. When the loss moved to a different link, these receivers were caught with highly inappropriate timer values.

5.9.2. SRM Trade-offs

The most significant trade-off in SRM is trading latency for implosion. The larger the back-off timers, the better the probability requests and replies will be suppressed. However, given that loss may randomly move around, it is very hard to determine the optimal value of the adaptive timers do that loss is always minimal. In addition, setting up the RTT state to each member takes time, and in dynamic groups this state may never converge to a stable state.

5.10. PGM Experiments

To the best of our knowledge, our simulation was the first simulation ever written for PGM. Results from our simulation were first presented at the 4th Reliable Multicast Research Group meeting [56], where one of the PGM designers from Cisco was present. There, we were informed that PGM was already at an advanced stage of implementation.

The topologies and parameters used for the PGM experiments are the same as with LMS and SRM, except that we did not have the memory limitations we experienced with SRM (the PGM simulation was written mostly in C++) and thus our simulations used 100 receivers, as in LMS. Our PGM simulation did not include all the features described in the PGM specification. However, we believe our simulation included enough of the PGM functionality to capture the basic operation of PGM. Thus, we believe that the results we present here are applicable to a full-fledged PGM implementation.

Figures 5.26, 5.27, and 5.28, show the PGM latency with loss at the source, receivers, and all links respectively. With loss at the source, average recovery takes about 80% of the unicast latency. One might wonder why this experiment did not produce results similar to LMS, since in both cases recovery is done from the source; the reason is due to PGM repeated retransmissions, which result

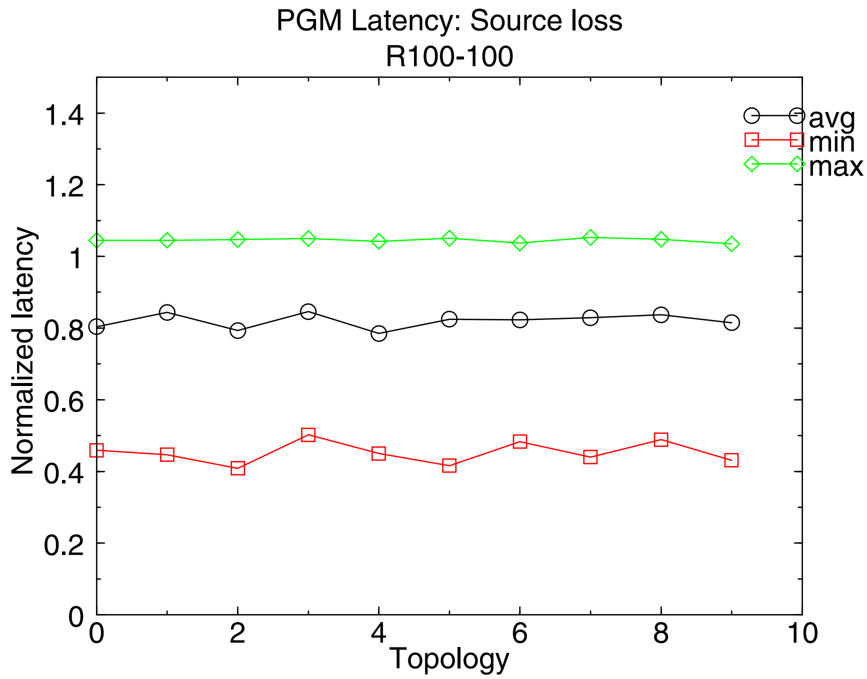

Figure 5.26: PGM Latency, loss at the source

in an increase of the average recovery latency. In addition, recall that in PGM receivers observe a random back-off before sending NACKs, which results in an increase in the overall recovery latency. We have used a back-off value of zero in our experiments; with larger back-off values, recovery latency will increase even more.

With loss at the receivers, the recovery latency is very close to 1, as expected. Note that with PGM recovery latency does not increase much beyond 1 because a retransmission always comes from the source. Thus, unlike LMS, PGM does not allow a receiver to send a retransmission which may result in a longer retransmission path. With loss at all links, recovery latency in PGM increases slightly (a trend also seen with LMS), to about 90% of unicast latency. This is about 30% more than what is seen with LMS. In summary, it appears that on average, allowing receivers to participate in recovery saves about 30 - 50% in recovery latency.

Figure 5.29 shows the PGM repeated retransmissions when loss occurs near the source. Recall that in PGM the source may send repeated retransmissions in response to the same packet loss if the RTT between receivers is such that a retransmission arrives before NACKs from downstream

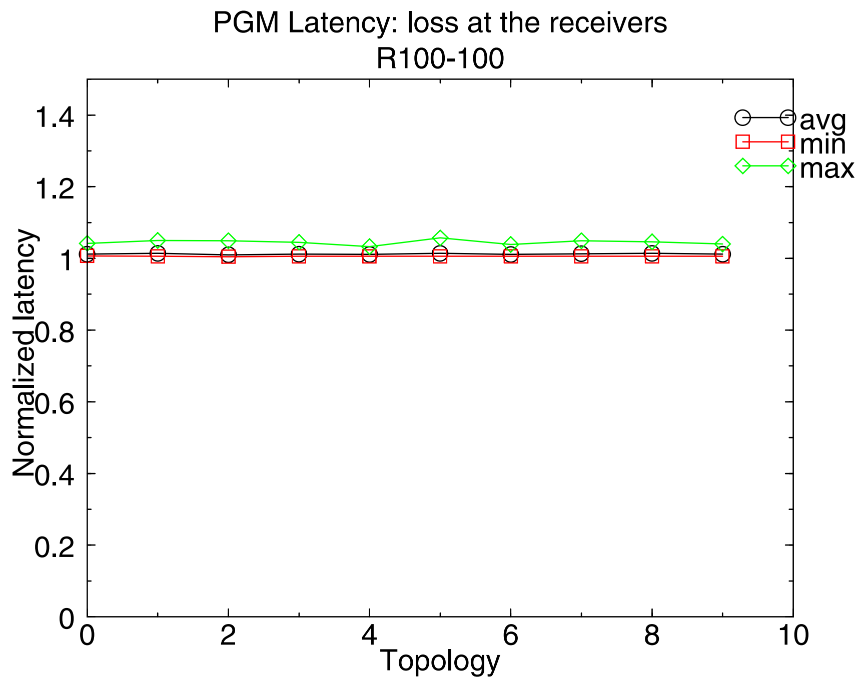

Figure 5.27: PGM recovery latency with loss at each receiver

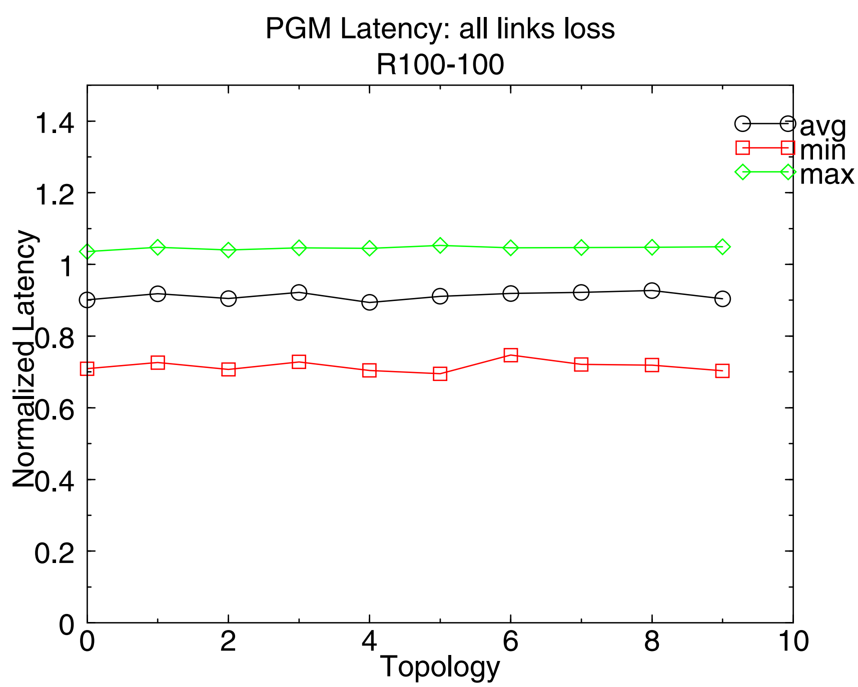

Figure 5.28: PGM recovery latency with loss at all links

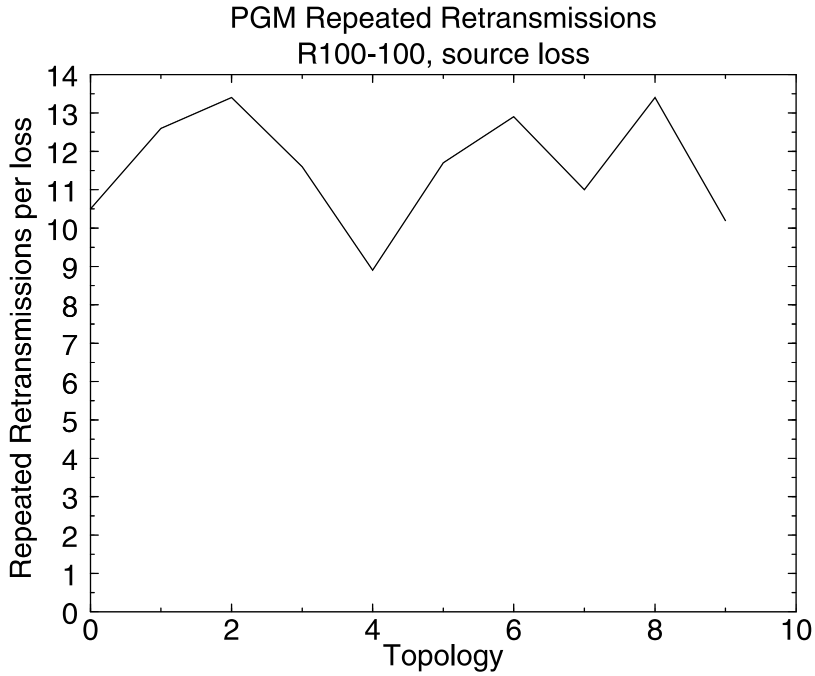

Figure 5.29: PGM repeated retransmissions with loss at the source

receivers at a particular router, wiping out the current retransmission state at the router. Then, when other NACKs arrive, they recreate the state all the way back to the source and trigger another retransmission. Loss that occurs near the source will create the maximum number of repeated retransmissions. Our results show that the number of repeated retransmissions can be quite high, between 9 and 13 retransmissions for each lost packet. This will invariably create a significant load at the source and may contribute to congestion.

This experiment did not include any back-off at the source, because it was not part of the initial PGM specification. Since then, the PGM designers have added a heuristic to reduce this problem, where the source delays the sending of a retransmission to allow NACKs to establish state at the routers. The difficulty in designing such a heuristic is how to determine the appropriate amount of back-off to minimize repeated retransmissions while not unduly increasing recovery latency.

Figure 5.30, shows the repeated retransmissions problem with loss distributed at all links. It is clear that the problem becomes much less pronounced in this experiment. It appears that the number of repeated retransmissions reduces by two orders of magnitude. Thus is hard to estimate the real-life effect of repeated retransmissions in PGM until we have a reasonably good loss model. What

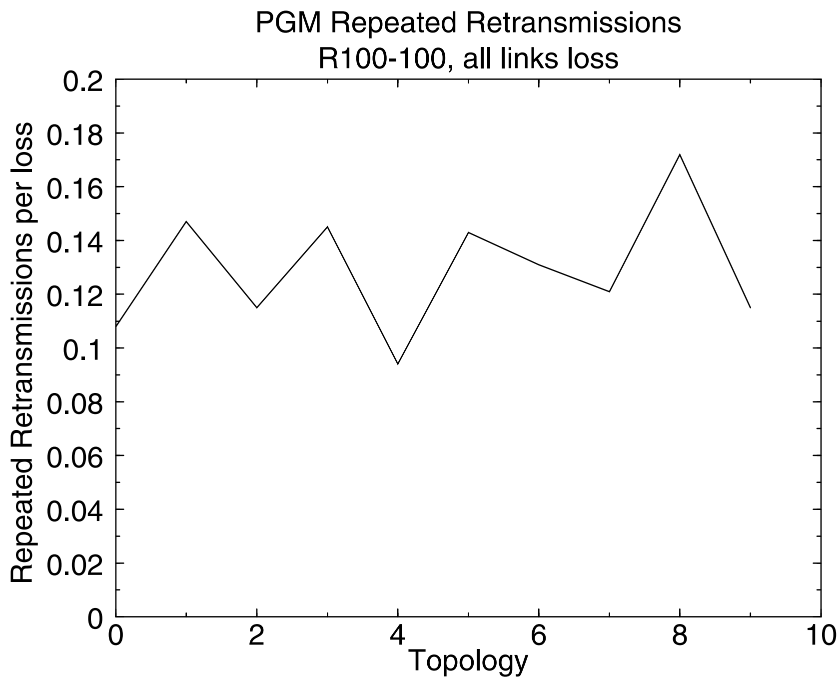

Figure 5.30: PGM repeated retransmissions with loss at all links

this experiment clearly demonstrates again is the sensitivity of multicast protocols to topology and loss location.

We do not present experiments measuring repeated retransmissions with loss at the receivers, because, clearly, there are no repeated retransmissions in this case.

5.10.1. PGM Trade-offs

PGM trades state at the routers for eliminating exposure and keeping recovery latency low. The state at the routers is proportional to the number of lost packets, and thus may increase significantly during periods of high loss. In addition, PGM needs to trade latency for eliminating repeated retransmissions, in cases where loss occurs near the source.

5.11. Summary and Discussion

In this chapter we presented simulation results measuring the performance of LMS, SRM and PGM. We used the *ns* simulator, which comes with SRM, and added implementations for LMS and PGM. The experiments included topologies generated via GT-ITM, a generator that generates topologies that approximate Internet topologies. We used the same topologies to evaluate all three schemes. We measured recovery latency for all three schemes, the exposure and number of duplicates for LMS and SRM, and number of repeated retransmissions for PGM.

In general, LMS and PGM perform much better than SRM, due to the assistance they receive from the routers. Improvements can be seen in both recovery latency, exposure and duplicates. We believe that these performance advantages alone, are important enough to warrant serious consideration of their deployment; for example, their lower latency may allow the implementation of multicast error control for continuous media applications; and their limited or absent problems with exposure, are important assets for scalability. These two schemes have the further advantage of freeing receivers from maintaining any topology related state and performing any topology related operations. This is an important advantage in terms of reducing application complexity, and will greatly ease the deployment of multicast applications.

Comparing LMS and PGM, we note that while LMS is much simpler to implement and requires significantly less overhead than PGM, its performance is far from lacking. LMS has significantly lower recovery latency, while trading very little in terms of exposure. PGM incurs per-lost-packet penalties at the routers, which for large routers (like those on the backbone) it can be significant. In contrast, LMS incurs only a small fixed state penalty per multicast group at the routers. Thus we believe that in terms of scalability, LMS has the edge.

The performance results are summarized in Table 5.1. Our results show that LMS has the fastest

Table 5.1: Performance summary of all three schemes

SCHEME	Normalized Latency (% of unicast latency)	Exposure/duplicates	Repeated retransmissions
LMS	30 - 60%	0.5%	none
SRM	>200%	4-6 duplicates per loss	none
PGM	80 - 100%	none	up to 10 - 13

recovery: a receiver typically recovers from loss in about 30-60% of its unicast latency to the source.

PGM, which always recovers from the source is next, requiring about 80-90% of the unicast latency. Thus it seams that the penalty PGM pays in terms of latency by not allowing local recovery is about 30-50%. SRM, due to its duplicate suppression mechanism, requires the longest time to recover, which is at least twice the unicast latency.

In terms of duplicates, SRM generates about 4-5 duplicate packets per loss. Without some form of local recovery, these duplicates will be sent to all receivers. LMS and PGM only allow one retransmission to reach a particular receiver. However, whereas PGM will never allow a retransmission to reach a receiver that has not sent a NACK, LMS may allow unwanted retransmissions to reach receivers that do not need them. Our simulations show that the probability of a receiver receiving unwanted packets is not very high, even with static repliers: typically, under 0.5% of all generated retransmissions will reach receivers that did not need them. PGM, while preventing unwanted packets at the receivers, may force the source to send repeated retransmissions in response to a single packet loss. The number of these retransmissions can be high enough (about 10 - 13 per loss) to be of great concern; however, without a better understanding of loss characteristics on the MBONE, it is hard to quantify the overhead of repeated retransmissions. What is clear though, is that PGM will have to incorporate some delay before sending retransmissions, which will increase its recovery latency beyond what we have reported here.