A Centralized, Tree-Based Approach to Network Repair Service for Multicast Streaming Media

Dan Rubenstein, Nicholas F. Maxemchuk, David Shur

AT&T Technical Memorandum TM HA1720000-991129-03
November, 1999

1 Introduction

The Internet's flexible design enables it to transmit a variety of types of traffic. Traffic can be as simple as a point-to-point data communication, or as complex as a large-scale, multimedia collaboration that simultaneously transmits information between numerous hosts. The deployment of IP Multicast [9] significantly reduces the load imposed on the Internet by large-scale, multi-host communications. However, IP Multicast is best-effort: a packet transmitted via IP Multicast need not reach all intended receivers. Hence, on its own, IP Multicast is unsuitable for sessions that require low packet loss rates.

Multimedia broadcast sessions, such as live or pre-recorded broadcast radio / TV, are applications that, via IP Multicast, can be made widely available to consumers at a low cost. However, peoples' interest in tuning in to such sessions decreases with the quality if the reception. Since packet loss degrades reception quality, it is necessary to add some additional mechanism on top of the best-effort IP Multicast service to keep loss rates at low levels.

Here, we present a system that improves reception quality by recovering from packet losses in streams transmitted via IP Multicast. The system utilizes a few dozen servers inside the network as *repair servers*. Each server caches packets from the original transmission, and can forward (via unicast) packets to other repair servers that did not receive the packets via the original transmission. After a fixed delay on the order of several seconds, a repair server then multicasts the packets in scoped sub-regions of the network, providing receivers within the scoped region with a transmission that is delayed by a few seconds relative to the original multicast session. However, this delayed transmission contains fewer losses, and thus presents a higher quality copy of the original transmission than what the receiver would have obtained had it joined the original transmission group.

We focus on producing a recovery system that can easily be deployed by a network service provider or by a set of cooperating network service providers. The system has several unique features. First, it is built to support existing and future applications that are designed to interact with the current best-effort IP multicast model. Our system does not require that these applications possess any additional functionality to handle late packet arrivals or to request repairs. Second, the system is fault tolerant: it uses a simple algorithm to route repair transmissions around any repair server that ceases to operate, and reconfigures itself to avoid subsequent failures. Last, the system's configuration information is maintained at a central point of control, simplifying system operation, monitoring, and on-line debugging.

There has been a significant amount of prior work that uses repair servers in a similar manner [1, 2, 3, 4, 5]. What separates our work from prior art is our focus on keeping the architecture simple where it doesn't need to be complicated. Doing so increases the likelihood of a successful deployment, and also simplifies the task of monitoring and debugging. This simplicity of the architecture is captured in terms of three basic design principles:

1. Decisions are centralized, as long as the centralization does not compromise the scalability of the system, and as long as the system can operate correctly and remain productive for an extended time if separated from the centralized decision point.

2. The complexity of the protocol at the repair servers is kept simple, and the amount of state that a repair server must maintain is kept low.

3. The system utilizes a simple model of the underlying network. A system designed to utilize a complex network model tends to be quite complex itself, and makes it difficult to assess its performance, or debug problems that may arise.

This paper specifically focuses on the protocol that is utilized by the repair system to repair packet losses at the repair servers, allowing them to deliver high quality multimedia multicast sessions to receivers. Our protocol provides high *reliability*: a high likelihood that a repair server can deliver a packet to a receiver within the receiver application's deadline. This is accomplished using a small amount of additional bandwidth on network links, keeping the operating cost low (in terms of link usage). In the protocol, repair servers obtain missing packets from other repair servers. What affects the reliability and cost of the protocol is the choice of *repair graph*: a graph whose nodes are the repair servers, and whose edges indicate the directions in which repairs flow between repair servers. We demonstrate how an algorithm, based on the algorithm that generates a minimum spanning tree, builds repair graphs that can achieve high reliability at a low cost. The only knowledge of the underlying network needed by these algorithms is the Euclidean distance between all pairs of repair servers, and between each repair server and the originating multicast source. We demonstrate through simulation that, given the limited knowledge we have about the underlying network, the repair graph generated by the algorithm achieves a level of reliability that is almost as high as that achieved by repair graphs specifically designed for high reliability. At the same time, it maintains a cost that is almost as low as the cost maintained by the repair graphs designed to maintain a low cost.

The paper proceeds as follows. In Section 2, we introduce the network model: our abstract view of the network and of the capabilities of the repair servers for which we wish to provide repair service. Section 3 presents the algorithms we use to generate the repair tree, and motivates why we believe these algorithms produce the kinds of repair graphs that would be most effective at providing high reliability at a low cost. Section 4 presents performance results of simulations of our repair graph algorithm on the network model. We discuss related work in Section 5, and last we discuss future directions and conclude in Section 6.

2 Network Model

In this section, we present our abstract model of the network on top of which we will be building a repair graph. We begin by describing the application model that we support, and then describe our abstraction of the Internet topology that we believe is a reasonable basis on which we should design our repair graph algorithm.

2.1 Application Model

Our repair graph algorithm is designed to connect repair servers that support multimedia broadcast applications, where the applications are designed to run on top of best-effort IP multicast. As a result, the repair server cannot rely on the application to actively participate in the repair service protocol (e.g., repair requests, playout buffering). Our model of the interface between repair server and application is based on the model presented in [6], which we now summarize.

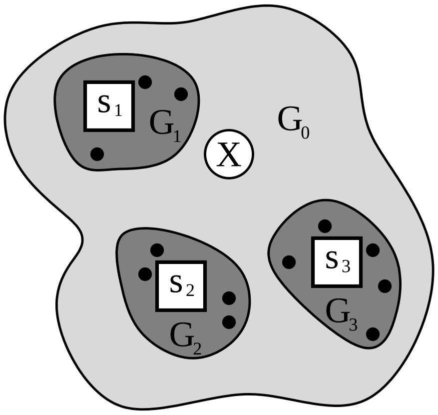


Figure 1: The Non-modified Application Repair Server Architecture

Figure 1 depicts how multicast groups are used by the repair system to improve reception quality at receivers. The transmission source, represented by an 'X', transmits its signal on a multicast group, G_0. Transmissions from the source onto G_0 have a large scope (e.g., a ttl of 127), and can be received in the lightly-shaded region. Some of the data generated by the source is lost within the network as it propagates to receivers (black dots) or repair servers (square white boxes) joined to multicast group G_0. Hence, receivers that join to G_0 will often receive a poor quality signal due to substantial packet losses.

Each repair server, s_i, joins to group G_0, and delays its transmission of the received data by some fixed amount of time, t_i, beyond its scheduled playback time. During this period of time, it detects packet losses in its received stream on group G_0, and attempts to recover these lost packets from neighboring repair servers via unicast requests. The recovery attempt consists of a series of one or more unicast requests to nearby repair servers, which unicast back the requested packet if it is available. When a time of t_i has elapsed beyond the playback time of a received or recovered packet, the packet is transmitted by the repair server, s_i, on a separate multicast group, G_i. Transmissions to G_i by s_i are scoped to within a local region (e.g., ttl of 0,1 or slightly larger). In Figure 1, we indicate the smaller scoping of the group G_i by a darker shaded region which surrounds the repair server.

Rather than join group G_0, a receiver that lies within the transmission range of some repair server, s_i, can join to group G_i and receive a delayed, but much higher quality transmission. This is because a receiver near a repair server s_i that is joined to group G_i receives almost all the packets that were received or recovered by s_i. If the repair graph is configured in a reasonable manner, then it is highly likely that a repair server s_i will be able to recover a packet as long as some repair server upstream on the graph receives the packet in the original transmission.[1] The repair will propagate downstream due to the chain of unicast requests and retransmissions of the lost packet along each hop of the repair graph. Thus, the likelihood that a receiver will receive a packet on G_i that it would have lost on G_0 can be extremely high.

Note that because the source and receiver do not participate in the recovery process, applications that have been and are being developed to use the standard IP multicast UDP-like interface can utilize the protocol by simply joining a repair multicast group instead of joining the original multicast group. Further details can be found in [6].

Repair servers do not maintain state about who has requested a repair. If a repair server, s_i, receives a request for a packet from a repair server, s_j, and it cannot provide the packet, s_i makes no response, and keeps no information regarding the request. Hence, it is s_j's responsibility to re-query s_i at some future point in time if it still desires to receive the packet from s_i. If each repair server were to maintain state about queries, it would be possible to propagate a repair through a chain of servers that lost the initial transmission in much less time. However, maintaining such state increases the complexity of the repair server code (which requires a re-request mechanism in any event because requests themselves might be lost), and is not necessary for the range of applications that we wish to support with our protocol.

In our system, the organization of the repair tree is formulated from a centralized process, which then contacts each repair server to inform it of its connection information. This centralized process assigns a set of parents to each repair server from which the server may request packet repairs. One can construct the repair graph from this information by drawing edges from a each repair server to its set of parents. The centralized process can contact servers on occasions when the repair graph needs to be updated (e.g., a node failure). However, we expect such updates to be rare. We have already built a repair system that operates

[1] A node A is *upstream* from another node B if there is a directed path from A to B.

in this manner, and have found it to be scalable. Also, it is easy to locate and route around faults in the network. Furthermore, repair servers whose services are not needed (no receivers are joined to the repair session, and no repair servers are requesting repairs) can enter a sleeping mode that conserves resources at the host running the repair server process, as well as in the neighboring network region. The description of these attributes of the system is discussed in [7].

2.2 Abstraction of the Underlying Multicast Network

Our abstraction of the underlying multicast network attempts to capture several realistic features of Internet multicast routing, and at the same time, avoid unnecessary details that we expect will not affect protocol performance. Since we compare the performance of repair graphs whose nodes consist of the transmission source and repair servers, our underlying network graph only contains nodes that can potentially serve as repair servers. We do this rather than introduce additional complexity in the model by including intermediate nodes that connect repair servers. We expect that removing these additional nodes does not significantly alter our results. Our model also does not include the last hop to the receivers that connect to the repair servers. Because one of our requirements is that application-level code cannot be modified to request or detect repair packets, there is little that can be done to improve the communication performance between a receiver and its "nearest repair server". We leave it to the receiving application (or user of the application) to choose a group on which it obtains the best service it can, given the limitations on the number of repair servers that can be deployed.

2.2.1 Underlying topology

Network nodes, each of which represents a potential source or repair server, are placed on a two-dimensional grid in which only a fixed subset of grid squares can contain nodes. The grid squares which contain nodes represent populated areas, such as cities, or, if one prefers a larger scale, continents. The grid squares that do not contain nodes represent unpopulated regions or oceans. We use a scaled down version of what is used in [8] to generate our graphs. In [8], up to 10,000 receivers are used per sample. Here, we are only interested in the placement of repair servers, and expect on the order of 50 nodes to be sufficient for providing a global repair service. We randomly select 5 grid squares from a 5x5 square grid to be populated regions. The remaining squares are oceans (devoid of nodes). We refer to each square that contains nodes as a *continent*.

After placing the nodes within the network on the continents, we randomly assign bi-directional links to connect the nodes, where the probability of constructing a given link is a function that decreases with the distance between the nodes it is to connect. Thus, nodes that are close together are more likely to be directly connected (it follows that the density of connectivity is higher within a continent than across continents). The algorithm to generate links does not terminate until a path exists between all pairs of nodes.

2.2.2 Building the multicast tree

Once we have generated the underlying network topology, we choose one node to be the source of the multicast group, and 20 of the remaining 49 unused nodes to be repair servers that will participate in the session. The multicast tree is the shortest-path tree from the source to each of the repair servers, where a path can proceed through any node within the underlying graph (whether or not the node has been selected as a repair server). We could perhaps build trees that are more realistic by increasing the aggregate number of nodes in the graph beyond 50. Doing so would likely increase the expected hop-count between repair servers. However, we expect that ISPs will choose repair servers in a strategic manner, and the shortest path between two nearby repair servers is likely to closely approximate the Euclidean distance between them.

2.2.3 Loss and recovery

We represent packet loss on the multicast tree as a Bernoulli process on each link of the tree. A repair server fails to receive a packet whenever any link upstream from it (toward the source) drops a packet. This process captures spatial, but not temporal, loss correlation observed for multicast transmission. In this paper, we will examine the system's ability to recover a single packet that is lost in parts of the network. For this reason, the fact that we do not capture the temporal loss correlations between successive transmissions is irrelevant.

For the purposes of building the repair graph, we permit each repair server to connect directly to any other repair server (i.e., the graph in which the 20 repair servers are the nodes and the edges represent paths for direct communication is a 20-clique). Again we make the assumption that the distance along the communication path between two repair servers closely approximates the Euclidean distance. Constructing the graph as such also implies that it is quite possible for two servers to have a more direct mode for communication than the path that connects them within the multicast tree.

We are interested in testing the performance of our system under two types of losses on top of the repair graph. First, we assume that the combined request and repair transmissions between two repair servers failing to deliver the repair from the requestee to the requester is a Bernoulli loss process. Second, we assume that it is possible that a small number of repair servers (0, 1, or 2) may fail during the session, so that they do not respond at all to requests for repairs.

2.2.4 Repair server algorithm

A repair server that detects a missing packet requests a repair immediately, and continues to do so periodically until the packet is received, or the packet's deadline for playout on the locally scoped multicast group expires. In the current implementation, the period of the request is a second. Since all repair servers are likely to detect a packet loss at approximately the same time (relative to a second), the ith request from all servers that have not received the packet occur at approximately the same time, and hence it makes sense to model the request process as a series of *rounds*. In each round, a repair server that has not yet recovered the packet makes a request to a parent for the packet. Because repair servers do not maintain state for packet requests, a repair propagates at most one hop on the repair graph per round toward a repair server that needs the packet.

3 A Repair-Graph Building Algorithm

Let us quickly review the traits of a “good” repair graph. We define the *reliability* of a repair graph more formally as the expected number of packets, averaged over all repair servers, that can be recovered before the deadline. We want a repair graph that exhibits a high reliability, close to 1. At the same time, we want to limit our usage of network resources. Here we assume that the cost of a transmission between repair servers equals the Euclidean distance between those servers. This is not an unreasonable assumption. At present, leased-line providers charge by the mile.

We also impose several additional requirements based on observations of trials of our prototype implementation [7]. The algorithm should be robust in environments where there are losses of repair packet transmissions, as well as over single node failures. Robustness over multiple node failures is of course preferred. However, it is unlikely that two nodes' times of failure will overlap (our system is capable of detecting node failures in under a minute [7], and in this same period of time it should be possible to restructure the repair graph to avoid the failed node). Last, we assume that the only information that is available to the algorithm is the geographical location of the source and of the repair servers in the network. Thus, we will not need to obtain accurate routing information, nor do we require initial estimates of the loss rates between various pairs of repair servers.

3.1 Toward a Robust Repair Graph

Before embarking on a description of our repair-graph building algorithm, we first describe at a higher level what motivated us to construct the algorithm the way we did.

3.1.1 A ring at the top

If the data source for a session participates in the repair protocol, then ensuring that a repair server eventually receives a copy of a lost packet (barring node failure, a flushing of the source's cache, or a deadline expiring) is simply a matter of rooting the repair graph at the source (making the source upstream from all repair servers). However, our assumption that the application code cannot be modified does not allow us to include the data source in the repair graph. This means it is conceivable that in some instances, the original transmission of a packet is lost by all repair servers, and is therefore unrecoverable. While we cannot hope to recover from such occurrences, we can try to construct a repair graph that gives a high likelihood of recovery in a small number of rounds in most loss scenarios by ensuring that a few hops upstream, there is a repair server that has a high likelihood of receiving the packet.

One method for locating such a repair server would be to observe the traffic for some period of time, and then perform a correlation study for each repair server to determine which server would best suit its needs for repair. However, such a process complicates the protocol, and with traffic patterns changing over time in the network, it is not clear that such a solution would be effective. Instead, we make the following observation: nodes with a smaller Euclidean distance to the source are more likely to lie upstream (nearer to the source) on the underlying multicast tree, and it follows that in most cases, it is more likely that these servers will receive the original data transmission, and subsequently be able to provide repairs.

Our solution is to identify a small set of repair servers that are near the source, where it is most likely that if any repair server receives a packet, then the packet was received by some repair server in this small set. We then connect the servers in the small set as a ring so that if any of them receive the packet, then all of them should receive the packet as the packet propagates around the ring. The remaining repair servers are connected to the ring from downstream, so that any packet that is received by some repair server within the ring can subsequently be obtained by any repair servers downstream.

3.1.2 Two Trees

A "good" repair graph efficiently distributes repairs from servers that are more likely to receive data in the initial transmission to those servers that are less likely to receive the data. Most often, a tree is constructed for this purpose, since it is the most efficient means of producing a fully connected graph (a path exists from the root of the tree to each node). However, a node failure, unless lying at a leaf of the tree, would partition the tree, and prevent propagation of repairs from nodes on one side of the partition to the other. To overcome this drawback, we need to construct a repair graph that has the efficiency close to that of a tree, but also has the ability to route around node failures.

3.2 Our solution

At a high level, our solution is the following:

- Select the 3 nodes that are likely to surround the source and construct a bidirectional ring from these nodes. Each node is connected to the other two. These three nodes are the root of our trees.
- Construct the best tree that we can from the remaining nodes.
- Remove the (directed) edges used in the primary tree, and generate a second tree rooted at the ring on the remaining edges. We would like the two trees to result in a graph that can route around any single node failure.

- During each round, a repair server that has not yet received a copy of the packet chooses an edge on one of its two trees and requests the repair.

We now proceed with the algorithm to construct the trees. We postpone our discussion of choosing the nodes from which we build the ring. For the time-being, the reader should assume that the ring has already been formed.

The first tree constructed is a modification of a minimum-spanning tree (MST). An MST is preferable because it is likely to reduce the aggregate cost of transmitting repairs. However, certain properties of an MST will decrease the reliability of the repair graph. Unlike a shortest path tree, the path to a repair server can be through a long chain of repair servers. Passing through multiple repair servers in this fashion can delay the receipt of a repair over many rounds. It may be advantageous to pay a slight penalty in cost and take a more direct but longer edge to an upstream node. Looking at this in terms of a network spread out over several continents, it makes sense to aggregate repairs to a small number of points before crossing oceans. However, nodes should reach this aggregation point in a small number of hops, and should not daisy-chain in order to achieve a small savings in cost.

3.2.1 Construction of the primary tree

We write $c(n, m)$ for the Euclidean distance from a node n to a node m, and measure the network cost for sending a repair over link (n, m) as being proportional to $c(n, m)$. In such a case, we say that m is the *parent* of n. We define $h_1(n)$ to equal the minimum number of hops it takes along in the primary tree to travel from a node in the ring at the root of the tree to node n. We measure the utility of the repair server located at node n as being inversely proportional to $h_1(n)$, i.e., utility increases as the minimum number of hops upstream to a node on the ring decreases.

1. Let N be the set of nodes without a path on the tree to the ring.
2. Let M be the set of nodes with a path on the tree to the ring.
3. While N is non-empty
4. Select $n \in N$ and $m \in M$ such that $c(n, m) + \alpha h_1(m) \leq c(n', m') + \alpha h_1(m')$ for all other $n' \in N, m' \in M$.
5. Add n to M, and add edge (n, m) to the graph.
6. end while

Figure 2: An iteration of the algorithm to build Tree 1

Figure 2 presents the main iteration within the body of the algorithm used to generate the primary tree. The three nodes that form the ring near the source are connected and placed in the set, M, of nodes which have a path on the tree to a node within the ring. N is initially the set of the remaining nodes. Each time the iteration of the body is performed, a node is moved from M to N. When M is empty, all nodes are attached to the tree and the algorithm is complete.

We now discuss the construction of the ring that forms the root of the repair tree. The objective is to try and "surround" the paths on the multicast tree from the source without underlying knowledge of the multicast tree. We want to keep small the number of nodes that we place within the ring to help keep the maximum hop-count to propagate a repair to all repair servers in the ring low. To accomplish these tasks, we run the algorithm to build the primary tree, rooted at the source, and then perform a breadth-first search on the tree. When two nodes have the same depth (in number of hops), we give precedence to the node with the smaller edge cost. The first three nodes we come across are the nodes that form the ring. In the first repair tree, the third node obtained in the breadth first search recovers from the second node, which recovers from the first, which recovers from the third.

3.2.2 Construction of the secondary tree

For the secondary tree, the ring at the root of the tree contains the same set of nodes as the ring that roots the primary tree, but the edge directions are reversed (i.e., the first node added to the ring recovers from the second, which recovers from the third, which recovers from the first). All directed edges used within the primary tree are removed from the underlying graph so that the secondary tree and primary tree share no common, directed edges. Note however that if directed edge (m, n) is in the primary tree, we permit the edge (n, m) in the secondary tree.

We define $h_2(n)$ to equal the minimum number of hops it takes in the secondary tree to travel from some node in the ring at the root of the tree to node n. The main purpose of the secondary tree is to ensure that the graph built from the combination of the two trees can route around any node failure. We accomplish this by restricting the set of edges that can be used in the secondary tree to those that satisfy certain depth requirements.

1. Let N be the set of nodes without a path on the tree to the ring.
2. Let M be the set of nodes with a path on the tree to the ring.
3. While N is non-empty
4. Select $n \in N$ and $m \in M$ where $h_1(n) \geq h_1(m)$, such that $c(n, m) + \alpha h_2(m) \leq c(n', m') + \alpha h_2(m')$ for all other $n' \in N, m' \in M$ where $h_1(n') \geq h_1(m')$.
5. Add n to M, and add edge (n, m) to the graph.
6. end while

Figure 3: An iteration of the algorithm to build Tree 2

Figure 3 presents the main iteration within the body of the algorithm used to generate the secondary tree. Note that this algorithm is the same as the algorithm in the first tree, except that a link (n, m) can be added to the tree only if it additionally satisfies $h_1(n) \geq h_1(m)$.

Theorem 1 *The primary tree algorithm and the secondary tree algorithm both produce trees rooted at the ring that connecting all nodes in the graph.*

Proof: If N is non-empty, then at least one node must satisfy the condition in step 4 of Figure 2. Hence, N is eventually exhausted of all nodes, so all nodes are connected. Aside from the nodes in the ring, each node is connected at most once to the tree, when it is moved from set N to set M, which ensures the resulting graph is a tree (minus the ring at the root).

For the secondary tree, consider any node n where $h_1(n) = i$. If $i = 0$, then the node is part of the ring and is trivially included in part of the graph. If $i > 0$, then there are always at least two nodes in the tree that the node can connect to (i.e., two nodes in the ring to which it has not previously joined). The remainder of the proof follows the proof for the primary tree. ∎

Lemma 1 *Let G be a graph consisting of all nodes in the graph from which our two trees were derived, and let the edges in G be the union of edges from the two trees. Then for any node m where $m \neq n$, there is a path from some node in the ring at the root of the tree to node n that does not proceed through m.*

Proof: We prove this by induction on the value of $h_1(n)$, the depth of n in the primary tree. If $h_1(n) = 0$, then the node is itself in the ring, and we are done. Assume the theorem is true for all depths $h_1(n) < d$, and now consider the case where $h_1(n) = d$. Let n' be the parent (the node immediately upstream) of n on the primary tree. Hence $h_1(n') = h_1(n) - 1 = d - 1$. If $n' \neq m$, then by the inductive assumption, there is a path from the ring to n' that does not proceed through m, and hence a path from the ring to n' to n that does not proceed through m.

We now only need to consider the case where $n' = m$. Then m is the parent of n, and $h_1(m) = h_1(n) - 1 = d - 1$. Consider the sequence of nodes $n_0, n_1, n_2, \cdots, n_k$ that connect node n to the ring in the secondary tree (each node n_i in the sequence satisfies $h_2(n_i) = i$, n_0 lies on the ring, $n_k = n$). If no $n_i = m$, then the path from the ring to n that excludes m is the path on the secondary tree, and we are done. Hence, assume some $n_i = m$. Since m is n's parent on the primary tree, then (n, m) was removed prior to the generation of the secondary tree and n must lie at least 2 hops upstream from n on the secondary tree, i.e., $i < k - 1$. The rule applied in step 4 of the algorithm in Figure 3 that builds the secondary tree ensures that $h_1(n_i) \leq h_1(n_{i+1}) \leq h_1(n_k)$, so we have $h_1(n) - 1 = h_1(m) = h_1(n_i) \leq h_1(n_{i+1}) \leq h_1(n_k) = h_1(n)$, so either $h_1(n_{i+1}) = h_1(m)$, or $h_1(n_{i+1}) = h_1(m) + 1$. Let n'' be the parent of n_{i+1} on the primary tree. Since $m = n_i$ is n_{i+1}'s parent on the secondary tree, m cannot also be n_{i+1}'s parent on the primary tree, so $m \neq n''$. Since $h_1(n'') = h_1(n_{i+1}) - 1 \leq h_1(m)$, m resides at a depth equal to or higher than the depth of n'' on the primary tree, and thus m cannot lie upstream from n'' on the primary tree. Thus, the path to n which does not proceed through m is the path from some node in the ring to n'' on the primary tree (unless n'' itself lies within the ring, in which case, the path starts at n''), and then proceeds to n_{i+1} (along the primary tree), and then follows the secondary tree via nodes $n_{i+1}, \cdots, n_k = n$. ∎

Corollary 1 *The repair graph derived from the union of the edges of the two trees can route around any single node failure.*

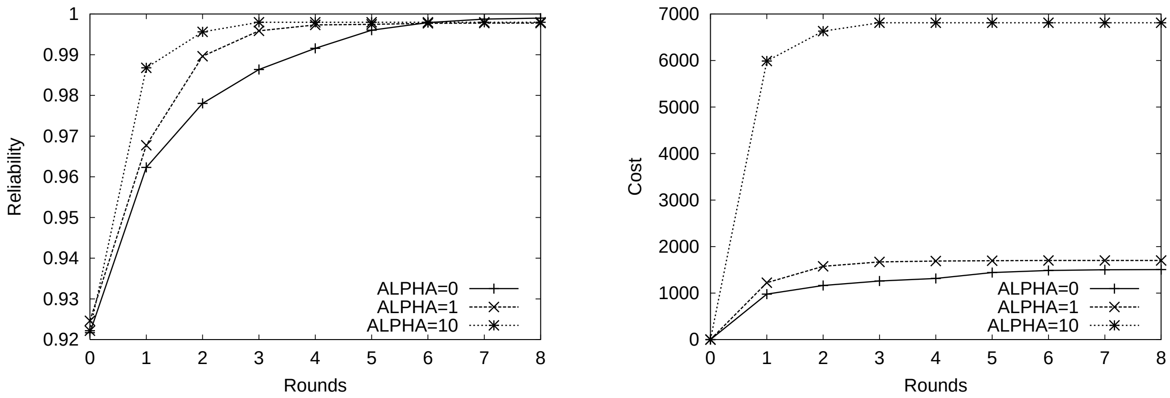
Proof: Choose any node m to fail, and by Lemma 1, there always remains a path from any node to the ring at the root avoiding m. ■

We conclude this section by pointing out that the algorithms for building the trees contain a tunable parameter, α. In the next section, we will examine how varying α impacts the reliability and cost of the resulting repair graph.

4 Evaluation

In this section, we evaluate the performance of our algorithm via simulation. Our evaluation proceeds in three steps. First, we determine an appropriate value for the tunable parameter, α, that achieves a low cost while maintaining a high likelihood of packet recovery (i.e., reliability) for all receivers. We will see that using a value of $\alpha = 1$ within the algorithm results in trees that use long, higher cost links infrequently (keeping cost low), and have a low hop-count for most repair servers to recover the packet (keeping reliability high). Next, we consider the impact of having a secondary tree in the repair graph. We find that utilizing the links on the secondary tree causes a small increase in cost and a small reduction in reliability in the case where there are no node failures, but that when node failures occur, requesting repairs from the secondary tree can increase reliability significantly. Last, we examine the impact on reliability and cost as we vary the loss rate on both the multicast tree and on the recovery tree. We find that the loss rate on the multicast tree reduces reliability of the repair graph, but this reduction applies to any generated repair graph. Hence, the repair graph generated with $\alpha = 1$ in which both repair trees are utilized in the repair process remains the preferred option. We find that loss on the repair graph has much less of an impact on both cost and reliability.

We run simulations as follows: A simulation configuration consists of a choice for α, a fixed loss rate on links of the multicast tree, a fixed loss rate on links of the repair tree, a fixed number (0, 1 or 2) of nodes that fail (are unable to forward repairs), and a round schedule that indicates per round the tree (primary or secondary) that a repair server should use to issue its repair request. For each configuration, we performed 500 runs. Each run consists of the following steps: First, we randomly generate a network, and randomly choose 21 out of the 50 nodes to be possible repair servers and multicast source. We then choose 10 different nodes at random to be the multicast source. For each choice of multicast source, we generate a multicast tree and repair graph (where the remaining 20 nodes chosen are repair servers). In the case where we have either one or two node failures, we iterate over all possible combinations of nodes failing, considering each combination once. We then transmit 10 packets, record how many repair servers were able to recover the packet by the end of each round, and what the cumulative cost of transmissions was at the end of each round. We average over all results, giving equal weight to each packet transmission in each run, to obtain the reliability and average cost.



(a) Reliability

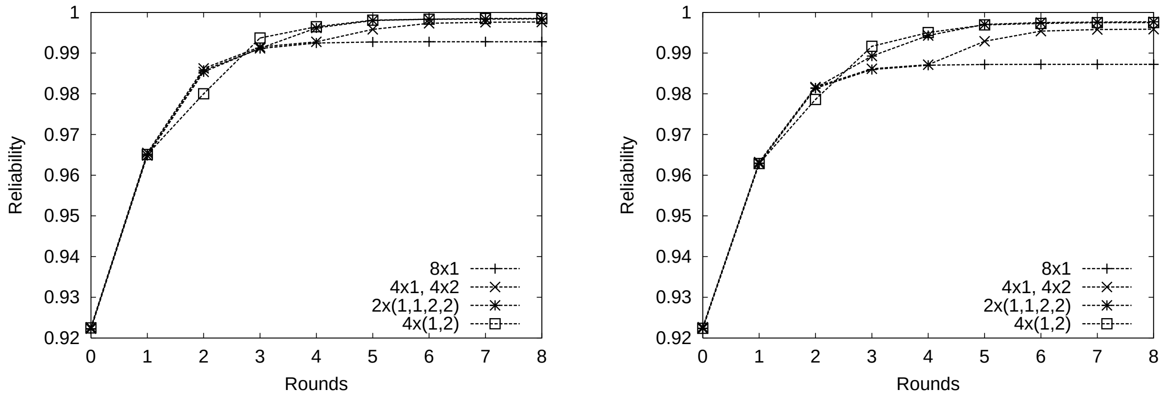
(b) Cost

Figure 4: How changing α changes reliability and cost

Figure 4 demonstrates how varying α impacts the reliability and cost of a repair session. Here, there is no loss of transmissions between repair servers, and there are no node failures. The loss rate on each link during the original multicast transmission of the packet is .03. Each repair server issues a request along the primary tree during the first four rounds, and switches to the secondary tree during the last four rounds, and gives up after eight rounds. On the x-axis of both Figures 4(a) and 4(b), we vary the number of rounds. The y-axis of Figure 4(a) indicates the average level of reliability, and the y-axis of Figure 4(b) indicates the average cost. Points plotted at $x = 0$ in Figure 4(a) indicate levels of reliability of the original multicast session (without any repairs). The various curves plot reliability and cost as a function of round for different values of α.

We see that by increasing α, we can increase the reliability of the repair graph. This is because a high value of α increases the weight of the hop-count component within the metric that the tree generating algorithm tries to minimize. As a result, the length of the path from a node in the ring at the root of the graph to the repair server is increased. This in turn increases both spatial correlation of loss and the maximum number of hops that a repair needs to travel in worst case scenarios. A value of $\alpha = 10$ is most often a star, with each repair server connecting directly to some node in the ring at the root. A value of $\alpha = 0$ produces a minimum spanning tree (if we consider the ring at the root as a single node before applying the MST algorithm).

We see a significant increase in reliability as we vary α from 0 to 1, and a significant increase as well when it is varied from 1 to 10. However, the increase in cost as we vary α from 0 to 1 is not nearly as significant as it is when it is varied from 1 to 10. We conclude that it makes the most sense to choose a value of α close to 1, since this gives a significantly higher level of reliability than that for $\alpha = 0$ without a significant increase in cost.



(a) One node failure

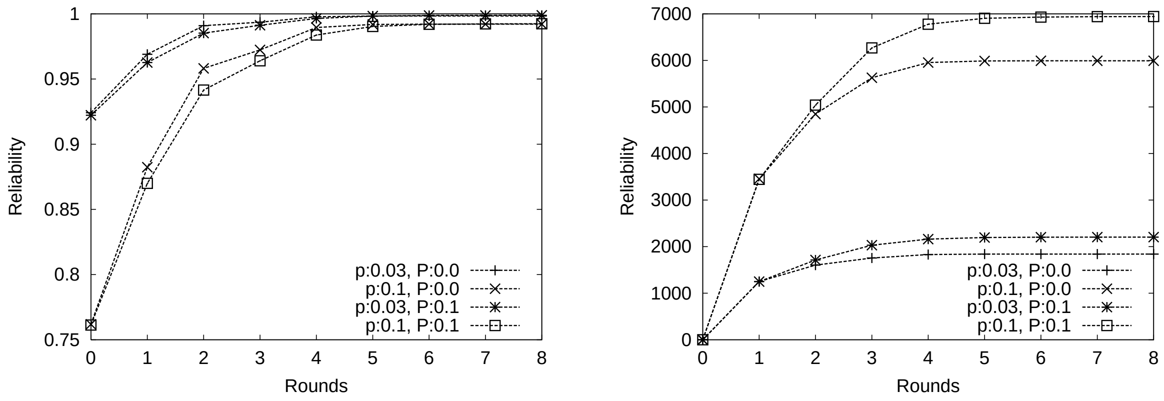
(b) Two node failures

Figure 5: The second tree's impact on node failures

Figure 5 demonstrates how utilizing a second tree can increase reliability in the event of node failures. Because our simulations use identical loss rates on all links, one does not achieve any significant gain in reliability by utilizing a second graph. However, using a single repair tree, a node failure might partition a set of repair servers that lost a packet from the set of nodes that might be able to repair the loss. Figure 5(a) demonstrates the impact that a single node failure has (recall we average over all possible configurations of the location of the node failure), and Figure 5(b) demonstrates the impact of two node failures at once. Again we vary the number of rounds on the x-axis and the reliability on the y-axis. The different curves represent different orderings of repair graphs used per round. For instance, the curve labeled "4x(1,2)" means that on odd rounds, the parent is chosen from the primary tree, and from the secondary tree on even rounds. We see that switching to the secondary tree allows a higher level of reliability, and this is exemplified in the unlikely event that there is more than a single node failure at any given point in time.

Figure 6 demonstrates the effect of varying the loss rate on the original multicast tree (labeled p) and on the repair tree (labeled P). We see that varying the loss rate on the original multicast tree has a larger impact on both the reliability and cost of the repair server system than similar variations in loss rate on the repair transmissions. This is not surprising, since increasing the loss rate on the original multicast tree increases the expected number of receivers that need additional repairs. Increasing the loss rate on the repair tree only affects the reliability of the subset of receivers that did not receive the initial transmission, and therefore this has a smaller impact on overall reliability.

Last, we find that the conclusions drawn by examination of Figures 4 and 5 remain the same for the loss rates depicted in Figure 6. In other words, we find that for reasonable loss rates, a value of $\alpha = 1$ and alternating parents between the two repair trees over the various rounds provides a repair system that exhibits the most desirable tradeoff between low cost and high reliability.



(a) Reliability

(b) Cost

Figure 6: The impact of loss on reliability and cost

5 Related Work

IP multicast [9] is designed as a best-effort service, shifting the responsibility of providing some sort of reliable delivery to the protocol layers that lie above the network layer. Several early works examine reliable multicast in networks where no explicit knowledge of participant location was necessary [10, 11]. However, the current preferred means to achieve a reliable multicast that scales to a large number of receivers is to form some sort of hierarchical structure in the network. The idea was first developed into a protocol by Paul et al [2] by building the hierarchy from participating receivers.

Providing better-than-best-effort service for deadline-driven traffic, such as audio and video applications, was examined independently by Maxemchuk et al in [6], Xu et al in [1], and Lucas et al in [4]. In these works, receivers request repairs from other receivers. The goal is to repair as many lost packets before their deadlines as possible, rather than providing full reliability, as would be necessary for a data-transfer. Xu's work was extended in [5] to demonstrate the performance benefits of including *waypoints*: servers inside the network dedicated to improving protocol performance. In the application presented in that paper, waypoints act as repair servers, and reduce the repairing load on receivers in the session.

A variety of works [12, 13, 14, 1, 5] present different algorithms for generating the repair hierarchy that can be used to provide reliable multicast. All of these approaches build their repair graphs dynamically using distributed algorithms. A dynamic algorithm to build the repair graph can customize the graph precisely to the current networking conditions. The distributed nature allows the algorithm to scale to a large number of tree participants. However, our results indicate that a simple, static, centralized algorithm builds a repair graph that is sufficient for our needs, both in terms of reliability and scalability.

6 Future Directions and Conclusion

We have proposed and examined a simple, static, centralized algorithm to build a repair graph to provide resilient multicast support to real-time multicast sessions that can tolerate small playback delays. The simplicity of the algorithm is due to the high level of abstraction of our underlying network model, and because decisions that are made infrequently are centralized. We start by limiting the set of features in the network that we model to only those that we believe are important, and design an algorithm that provides a high level of reliability while maintaining a low link utilization cost for this simple model. The simplicity of the algorithm makes the system easy to deploy, and its static, centralized nature facilitates understanding and/or debugging of the protocol in a real Internet environment.

In the near future, we hope incorporate our algorithm into our existing repair service system and verify its effectiveness at providing repairs within the actual Internet. It would also be of interest to compare the level of reliability and link utilization cost of our approach with the protocols that build their repair graphs in a distributed, dynamic fashion to see whether the performance gains due to dynamic adaptation warrant the extra complication in the algorithms.

References

[1] R.X. Xu, A.C. Meyers, and H. Zhang. Resilient Multicast Support for Continuous Media Applications. In *Proceedings of NOSSDAV'97*, St. Louis, MO, July 1997.

[2] S. Paul, K.K. Sabnani, J.C. Lin, and S.Bhattacharyya. Reliable Multicast Transport Protocol (RMTP). *IEEE JSAC*, 15(3):407–421, April 1997.

[3] S. Kasera, J. Kurose, and D. Towsley. A Comparison of Server-Based and Receiver-Based Local Recovery Approaches for Scalable Reliable Multicast. In *Proceedings of INFOCOM'98*, San Francisco, CA, March 1998.

[4] M.T. Lucas, B.J. Dempsey, and A.C. Weaver. MESH: Distributed Error Recovery for Multimedia Streams in Wide-Area Multicast. In *Proceedings of IC3N'97*, 1997.

[5] K. Sripanidkulchai, A.C. Meyers, and H. Zhang. A Third-Party Value-Added Network Service Approach to Reliable Multicast. In *Proceedings of ACM SIGMETRICS'99*, Atlanta, GA, May 1999.

[6] N.F. Maxmchuk, K. Padmanabhan, and S. Lo. A Cooperative Packet Recovery Protocol for Multicast Video. In *Proceedings of ICNP'98*, Atlanta, GA, October 1997.

[7] N.F. Maxemchuk, Dan Rubenstein, and David Shur. Repair Server Activation Implementation. *Technical Report in preparation*, August 1999.

[8] N.F. Maxemchuk. Video Distribution on Multicast Networks. *IEEE Journal on Selected Areas in Communications*, 15(3):357–372, April 1997.

[9] S. Deering and D. Cheriton. Multicasting routing in datagram internetworks and extended LANs. *ACM Trans. on Computer Systems*, 8(2):85–110, May 1990.

[10] D. Towsley, J. Kurose, and S. Pingali. A Comparison of Sender-Initiated and Receiver-Initiated Reliable Multicast Protocols. *IEEE JSAC*, 15(3):398–406, April 1997.

[11] S. Floyd, V. Jacobson, C. Liu, S. McCanne, and L. Zhang. A Reliable Multicast Framework for Light-Weight Sessions and Application Level Framing. *IEEE/ACM Transactions on Networking*, 5(6):784–803, December 1997.

[12] M. Hofmann and M. Rohrmuller. Impact of Virtual Group Structure on Multicast Performance. In *Proceedings of 4th COST237 Workshop*, Barcelona, Spain, December 1997.

[13] B. Levine, D. Lavo, and J.J. Garcia-Luna-Aceves. The Case for Concurrent Reliable Multicasting Using Shared Ack Trees. In *Proceedings of ACM Multimedia'96*, Boston, MA, November 1996.

[14] B. N. Levine, S. Paul, and J.J. Garcia-Luna-Aceves. Organizing multicast receivers deterministically according to packet-loss correlation. In *Proc. Sixth ACM International Multimedia Conference (ACM Multimedia 98)*, Bristol, U.K., September 1998.